PSD813F1/80C31 Design Tutorial

Application Note 057

By Dan Harris and Mark Rootz

February, 1999

47280 Kato Road, Fremont, CA 94538
Telephone: (510)-656-5400
(800) TEAM-WSI (832-6974)
Web Site: http://waferscale.com
E-mail: info@wsipsd.com

Return to Main Menu

March 20, 1999—REV 1.2

PSD813F1/80C31 Design Tutorial

Application Note 057

By Dan Harris and Mark Rootz

Contents

1 Introduction 3
1.1 Design Example 5
1.2 Matching the Functions to a PSD813F1 10
2 The PSD813F Functional Blocks 11
3 PSDsoft Development Tools 13
3.1 PSDabel 13
3.2 PSD Configuration 13
3.3 PSD Fitter 13
3.4 PSD Simulator 14
3.5 Parallel Programmer 14
3.6 JTAG Programmer 14
3.7 C code Generation 14
4 Design Flow 15
4.1 PSDsoft Program Flow 15
5 813F Tutorial Example 17
5.1 Managing the Project 17
5.2 The PSDabel File 18
5.2.1 Compiling the Tutor Design 19
5.2.2 Simulating your design using ABEL simulation 21
5.3 PSDsoft Configuration 23
5.4 PSD Fitter : Fitting and Address Translation 26
5.4.1 Fitting the Design 26
5.4.2 Generating C code 28
5.4.3 Performing the Address Translation 30
5.5 PSD813F Chip Simulation 33
5.5.1 *PSDsoft.run* File 33
5.5.2 Running the Logic Simulator 34
5.5.3 Running the Analyzer 36
5.5.4 Working With the Explorer 37
5.6 Programming the PSD813F 38
5.6.1 PSDpro 38
5.6.2 JTAG—FlashLink 41
6 ISP and the PSD813F 53
Appendix A—ABEL Design File—*Tutor8xx.abl* 54
Appendix B—Stimulus File—*Tutor8xx.stl* 57
Appendix C—List of PSD813F Simulation Signals 61
Appendix D—Design file for EPM7064S (U2 of Figure 1) 63
Appendix E—Discrete Solution (Figure 1) Compared to Integrated PSD Solution (Figure 2) 66
Appendix F—System Memory Map and UART ISP 67

1 Introduction

This tutorial takes you step-by-step through the development cycle of a PSD813F based design, from design entry, to programming the device. The first part of this tutorial shows how a PSD813F1 can be used in conjunction with a handful of other ICs to implement an automatic gain control (AGC) design. The tutorial also shows how this design would be implemented using a discrete part solution, and Appendix E reveals the various benefits of using a PSD813F device versus the discrete solution.

The members of the PSD813F family of programmable system devices are Flash-based peripherals for use with embedded microcontrollers (MCUs), and are In-System-Programmable (ISP). These PSDs are designed to easily interface to a variety of 8-bit MCUs and provide them with memory, logic, and I/O.

Embedded designs are typically bound by cost, size, and power consumption. The market for products using embedded MCUs is extremely competitive. Time-to-market and quality features-per-dollar define success. Using a PSD813F device will reduce your:

- ✓ Cost
- ✓ Time-to-market
- ✓ Power consumption
- ✓ Board space
- ✓ Design complexity
- ✓ Chip count.

As you read this document, you will learn how the PSD813F can enhance your MCU, and meet its needs for:

- ✓ Flash memory
- ✓ EEPROM
- ✓ SRAM
- ✓ Configurable I/O pins
- ✓ Programmable logic (both sequential and combinatorial)
- ✓ Decoded address space
- ✓ Address expansion
- ✓ Backup power
- ✓ Code integrity
- ✓ Code security
- ✓ ISP.

All of these features in one cost-effective PSD813F1 device allow the use of a low cost, minimal feature, ROM-less MCU device.

In addition to giving step-by-step design entry information, this document highlights three areas:

1) ISP using concurrent memory or JTAG
2) Micro⇔Cell technology
3) The logic simulation capabilities of PSDsilosIII

A typical MCU design with Flash memory consists of the MCU, the main Flash memory, and either a boot PROM or SRAM to implement an ISP download to main Flash memory over a UART channel, or some other communication link. For systems that use SRAM for ISP, the Flash programming algorithm must first be downloaded to SRAM and then the MCU executes from SRAM during ISP. Any power interruption or system glitches that occur during the download may cripple the system. Therefore, a boot PROM is a necessity for applications that demand high system reliability. However, a boot PROM adds cost to the system and is difficult to update once in service. Flash-based PSDs address these concerns and combine all of the elements necessary to enable the MCU to easily download to main Flash memory and boot memory while in-system.

The ISP method just described requires MCU participation. The PSD813F also offers another ISP method that uses a JTAG interface, and requires no MCU participation. This means that a completely blank PSD can be soldered into place, and the entire chip can be programmed in-system using WSI's JTAG FlashLink cable and PSDsoft development software. This is a powerful new feature of the PSD813F that allows for easy field updates.

Typically, adding a peripheral to MCU memory space involves a great amount of circuitry to decode addresses lines, latch data, and handle bus timing. This "overhead" circuitry required to add a peripheral is not needed if a PSD813F device is used since the MCU address, data, and control signals are already routed and processed inside the PSD. Micro⇔Cells take advantage of this, allowing the designer to build logic peripherals inside the PSD in an efficient and flexible manner. This tutorial compares a PSD Micro⇔Cell design with an equivalent functional design using an Altera EPM7064S CPLD device to emphasize the efficiency of the PSD.

The PSD813F has 16 Output Micro⇔Cells (OMCs) and 24 Input Micro⇔Cells (IMCs). Each Micro⇔Cell occupies a memory location in the MCU address space and is connected to the data bus. The ability to load the flip-flops in the OMCs and read them back is useful in such applications as loadable counters, shift registers, and other system logic. The IMCs can latch external inputs and be read by the microcontroller. IMCs are also useful for implementing handshake communication logic with an outside source.

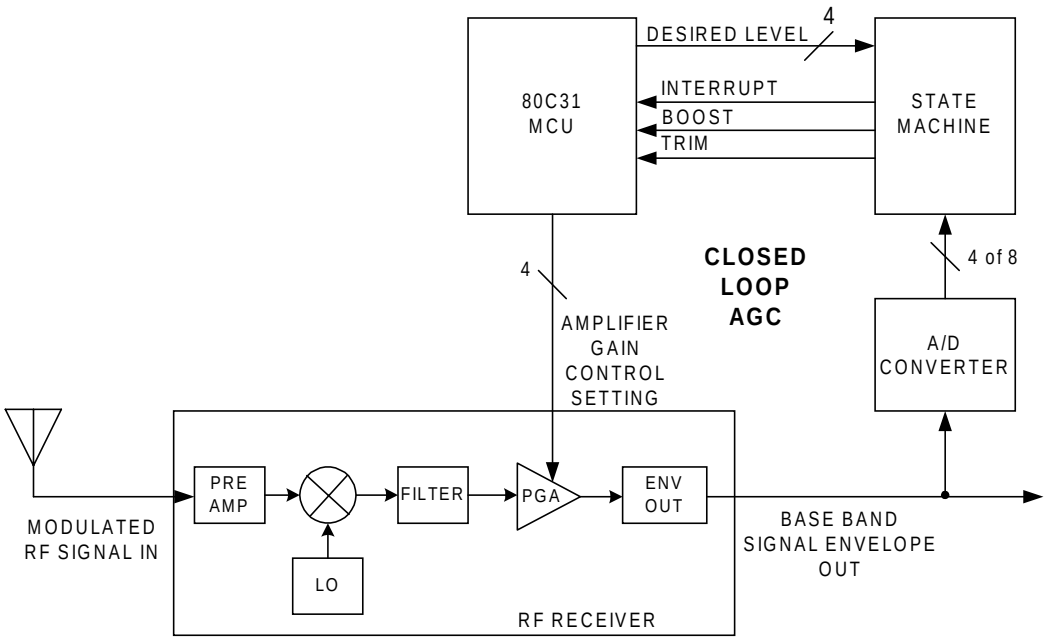
WSI provides complete chip-level Verilog-HDL models of all PSD devices for use with the optional PSDsilosIII simulator. These models can be used in conjunction with a user-defined stimulus file to simulate functionality of the PSD. PSDsilosIII also comes with a Waveform Editor/Viewer and Watch window (for stepping through the simulation) that are used in conjunction with the stimulus file. Most of the PSD's status and control signals, as well as all the user-defined logic in the CPLD, are available for use with the Waveform Editor/Viewer and the Watch window. Thus, the user can define MCU-level tasks, such as read and write that can be used as external chip-level stimuli to the PSD. The results of the stimuli can be viewed using the Waveform Editor/Viewer and Watch window of PSDsilosIII.

A new utility is featured in PSDsoft version 5.X. This utility automatically generates ANSI-C code for PSD functions, and can be used with the user's choice of MCU cross-compilers.

Note: The screen captures for this tutorial were taken using PSDsoft version 5.07. If you have a version prior to 5.07, you should update via our web site—http://waferscale.com. If you have a later version, some screens may look different, but the functionality should be the same.

1.1 Design Example

Implemented in this design example is a closed-loop Automatic Gain Control (AGC) function. An analog RF receiver section has a Programmable Gain Amplifier (PGA) to control the signal level that is output though an envelope detection circuit. The PGA gain must be adjusted in real-time to keep a constant signal level at the envelope detection output. This output is monitored by an Analog-to-Digital Converter (ADC). When the AGC function works properly, a constant signal level is output from the receiver. This signal can be used by other analog and digital circuitry for signal processing. The block diagram of the circuit is shown below.



Block Diagram of Automatic Gain Control circuit

The MCU could perform the real-time gain adjustment, leaving little MCU execution time for other tasks. Therefore, it is highly desirable to free up the MCU by off-loading these repetitive tasks with hardware. This is accomplished by moving some of this AGC functionality into a state machine in programmable logic.

In the above configuration, the MCU will load the state machine with a desired signal level and start the state machine. The MCU can then perform other tasks and can be interrupted by the state machine. The interrupt occurs if the signal drifts from the desired level. The state machine can do this because it is reading the outputs of the ADC and comparing the measured value with the desired value. The state machine will provide two additional signals with each interrupt, 'Trim' and 'Boost'. If the signal level from the receiver is too high, interrupts will be accompanied by 'Trim', and the MCU will decrement the gain value in the PGA. Likewise, if

the signal level is too low, interrupts from the state machine will be accompanied by 'Boost'. Once this closed-loop process is started, the MCU can perform other tasks and need only be interrupted when a gain correction is required.

This tutorial shows how to implement this AGC function two different ways: a discrete IC solution (individual IC devices for programmable logic, memory, etc.) and an integrated PSD solution. In addition to the AGC function, other features are implemented, such as a Real-Time-Clock (RTC), In-System Programmability (ISP), and miscellaneous I/O signals.

Please refer to Appendix F for information related to system memory mapping, ISP issues using a UART, and memory paging considerations.

We'll look at the discrete solution first, which requires four extra IC devices to implement the functionality described previously. It is an 80C31 MCU application that has a 128K x 8 Flash memory, a 2K x 8 battery-backed SRAM, a 32K x 8 EEPROM, a real-time clock (RTC), an 8-bit analog-to-digital converter (ADC), a JTAG interface, an EPM7064S ISP CPLD, and an analog receiver circuit (including PGA). Following that, we show how the Flash, EEPROM, SRAM, CPLD, and battery backup circuit are all combined into one PSD813F1 device.

The individual ICs of the discrete solution, shown in Figure 1, are described below.

U1 80C31 MCU—this MCU is using external memory since internal program and data storage is not sufficient. As a result, Port 0 and Port 1 are sacrificed for address and data.

U2 EPM7064SLC84-5 CPLD—needed for address decoding, control logic, implementation of a paging/segmentation scheme for the Flash and EEPROM, and interfacing to the PGA and ADC. Refer to Appendix D for the complete design listing for U6.

U3 29F010 Flash—128K x 8 program memory. Notice that address lines A14-A16 are driven by the CPLD to support additional address space.

U4 A128C256 EEPROM—32K x 8 boot memory. Allows concurrent programming of the Flash. Address lines A13-A14 are driven by the CPLD to support additional address space.

U5 DP8572A RTC—programmable Real-time Clock used to time-stamp various data received by the MCU.

U6 LH5116—2K x 8 SRAM configured with battery backup protection.

U7 Generic 8-bit ADC—converts the target signal envelope into a digital value. This IC is controlled by the CPLD.

U8 Receiver Circuit—collection of components that make up a signal receiver circuit, including: a Pre-Amp, a mixer, a Local Oscillator (LO), a PGA, and an Envelope Detector circuit. The circuit takes an RF signal through the antenna as input and outputs the signal envelope.

U9 7414 Inverter with hysteresis—U7B is used to provide a stable reset signal to the MCU (U1). U7A is part of the battery backup circuit for the SRAM.

U10 Generic OPAMP comparator—part of the battery backup circuit for the SRAM. When V_{cc} sags below the battery voltage, the circuit switches over to the battery, which then powers the SRAM.

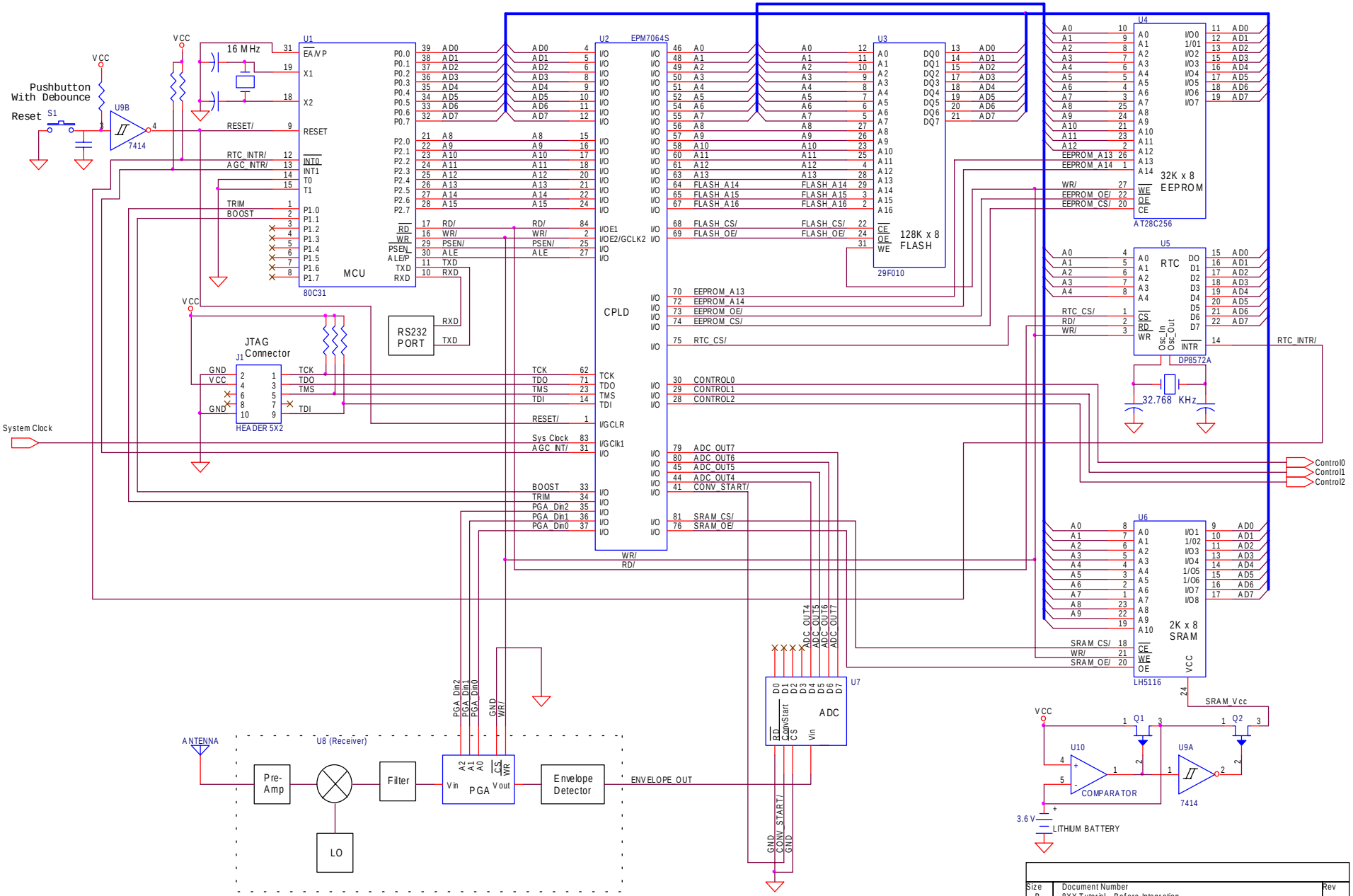
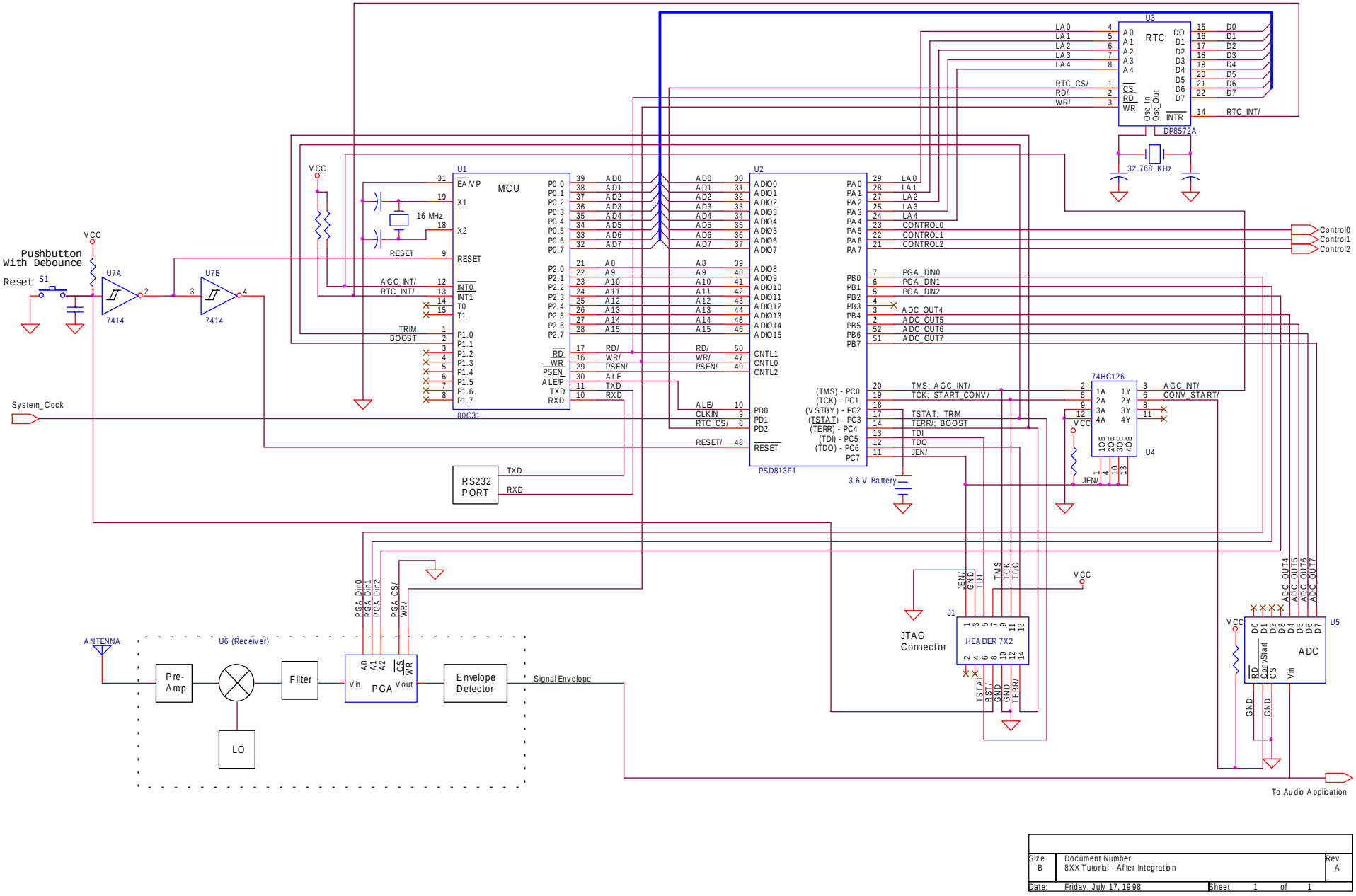

Pushbutton
With Debounce
Reset
S1
U9B
7414
VCC
16 MHz
U1
80C31
MCU
RESET/
RESET
RTC_INTR/
AGC_INTR/
INT0
INT1
T0
T1
TRIM
BOOST
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
RD
WR
PSEN
ALE/P
TXD
RXD
JTAG
Connector
J1
HEADER 5X2
GND
TCK
TDO
TMS
TDI
RS232
PORT
U2
EPM7064S
CPLD
RESET/
Sys Clock
AGC_INT/
BOOST
TRIM
PGA_Din2
PGA_Din1
PGA_Din0
SRAM_CS/
SRAM_OE/
CONV_START/
ADC_OUT4
ADC_OUT5
ADC_OUT6
ADC_OUT7
EEPROM_A13
EEPROM_A14
EEPROM_OE/
EEPROM_CS/
RTC_CS/
CONTROL0
CONTROL1
CONTROL2
FLASH_A14
FLASH_A15
FLASH_A16
FLASH_CS/
FLASH_OE/
U3
128K x 8
FLASH
29F010
U4
32K x 8
EEPROM
AT28C256
U5
RTC
DP8572A
32.768 KHz
Osc_In
Osc_Out
INTR
RTC_INTR/
U6
2K x 8
SRAM
LH5116
SRAM_Vcc
U7
ADC
ConvStart
Vin
U8 (Receiver)
ANTENNA
Pre-
Amp
LO
Filter
PGA
Vout
Envelope
Detector
ENVELOPE_OUT
U10
COMPARATOR
U9A
Q1
Q2
3.6 V
LITHIUM BATTERY
Control0
Control1
Control2
Size
B
Document Number
8XX Tutorial- -Before Integration
Rev
Date:
Friday, July 17, 1998
Sheet
1
of
1

Figure 1

Now, let's compare the integrated PSD design in Figure 2 to the discrete design in Figure 1. The memory (U3, U4, and U6) and the battery backup circuit (U9A and U10) of Figure 1 are all incorporated into the PSD813F1 (U2) of Figure 2. Also, all of the functions handled by the CPLD (U2 of Figure 1), are now taken care of by the PSD's CPLD. I/O pins are individually configured to match the functions implemented in the original design.

Using JTAG, the entire PSD813F1 device can be programmed. Also, the PSD JTAG pins can be multiplexed with other I/O. These JTAG features are beyond the capabilities of the EPM7064



Size	Document Number	Rev
B	8XX Tutorial - After Integration	A
Date: Friday, July 17, 1998	Sheet 1 of 1	

Figure 2

1.2 Matching the Functions to a PSD813F1

Table 1 maps the functional areas of the discrete solution into the PSD813F1.

The 80C31, running at 16 MHz, has a t_{aviv} (address valid to instruction valid time) of 207 ns. A PSD813F1-15 (150ns part) was selected to meet the 80C31 access time requirement.

Table 1. Discrete solution compared to the PSD813F1

Functional Area	Design Example with Discrete Components	Matching PSD813F1 Function
Memory	1. 128KB Flash	Same
	2. 32KB EEPROM	Same
	3. 2KB SRAM	Same
Memory Paging/ Segmentation, and Control	Extra logic to drive the address lines, output enables, and chip selects to the Flash and EEPROM	Automatically taken care of internally by the DPLD, PSD page register, PSD VM register, and prioritized memory access.
PLD/Control/ Demux	1. Decoder (EPM7064S)	Use DPLD (Decoding PLD)
	2. Address latch logic in CPLD	Port A in latched address mode (A7-A0)
	3. Various registers used to hold data or control information to be used by external devices	Use one Output Micro⇔Cell per bit for each register
I/O	1. Latched data inputs and outputs on CPLD	MCU I/O mode feature
	2. Combinatorial outputs on CPLD	same
Supervisory/ JTAG	1. Automatic switch to battery backup	Built-in comparator automatically switches to battery power when the system voltage drops below the battery voltage on pin PC2 (V_{stby})
	2. Limited JTAG interface with no multiplexing of the JTAG port available, and no JTAG ISP of memory available	Utilizes standard JTAG and non-standard extensions (to speed programming); the JTAG port can be multiplexed with other I/O, and the memory and logic within the PSD is ISP via the JTAG port.

2 The PSD813F Functional Blocks

The PSD813F provides five system-level functional blocks, and allows the user to define and configure these blocks to meet the design specification.

1. MCU Bus Interface

Adapts the address, data, and control lines of a particular MCU to the PSD. Choices include multiplexed or non-multiplexed address/data bus and the associated control/handshake signals.

2. PLDs (Decode for memory and registers, General logic)

The DPLD generates internal chip selects for the following PSD813F internal blocks:

- ✓ Flash memory
- ✓ EEPROM
- ✓ SRAM
- ✓ Control registers
- ✓ I/O Ports
- ✓ Peripheral I/O mode
- ✓ Micro⇔Cells.

The CPLD implements general logical functions, such as state machines, shift registers, counters, and combinatorial logic.

Both PLDs are based on Flash memory technology.

3. I/O Ports

The PSD813F has four I/O ports: Ports A, B, C, and D. These ports have several modes of operation and may be selected within PSDsoft during design entry or by MCU firmware at runtime. Modes that are defined by PSDsoft are implemented with Non-Volatile Memory (NVM) configuration bits that cannot be altered unless the device is reprogrammed. The remaining available I/O port operational modes are determined by the MCU writing to PSD control registers. See *Application Note 55* for more details.

4. Memory

The PSD813F1 has 128 Kbytes of Flash memory, 32 Kbytes of EEPROM, and 2 Kbytes of battery-backed SRAM. All of these memories may operate concurrently. That is to say that, while the MCU is executing code from one type of memory, the other memories may be written to, erased, or read.

These memory blocks are placed in system address space using PSDsoft development software. The PSD813F also offers some run-time features that can be used to alter the system memory map on-the-fly, which is good for memory paging and ISP.

5. JTAG ISC interface

The PSD813F family includes a JTAG channel for In-System Programming (ISP). This ISP function is an extension of the typical JTAG boundary-scan function. It is an implementation of the JTAG-ISC (In-System Configuration) specification that is becoming an industry standard. The entire PSD device may be configured and programmed while soldered to the end product. The PSD can be completely blank before programming because the JTAG interface needs no assistance from the MCU. WSI has enhanced the standard four-wire IEEE 1149.1 JTAG interface by making two additional handshake lines available to speed-up programming. Refer to *Application Note 54—JTAG Information - PSD8xxF* for more information.

The use of the JTAG interface and the two additional handshake lines are defined using PSDsoft. Also, the MCU has some control over the JTAG interface at runtime.

3 PSDsoft Development Tools

PSDsoft is WSI's integrated system development software tool, which runs on a PC in the Windows 95/98 and Windows NT environments. PSDsoft supports the configuration of the functional blocks described in the previous section. Figure 3 (of section 4) shows the PSDsoft design process flow for PSD devices. PSDsoft consists of the following major modules:

- **PSDabel**
- **PSD Configuration**
- **PSD Fitter**
- **PSD Simulator**
- **Parallel Programmer**
- **JTAG Programmer**
- **C Code Generator.**

Section 5 shows how each of these modules is used in conjunction with a PSD813F during a typical design cycle.

3.1 PSDabel

PSDabel has MINC's HDL ABEL (formerly DATA I/O ABEL) engine at its core. The PSDabel environment provides an editor to create/edit a .abl file that can be used to define chip select logic, general-purpose logic, and PSD configuration parameters. Template files are provided for many MCU and PSD combinations. When the .abl file is compiled, logic is synthesized and files are created and passed on to the PSDsoft fitting utility.

3.2 PSD Configuration

This utility is used to specify the PSD MCU bus interface type, special I/O pin assignments, and particular internal PSD functions. The output of this module is the .glc configuration file, which is also used by the PSDsoft Fitter.

3.3 PSD Fitter

PSD Fitter has two main functions: the Fitter and the Address Translator. The Fitter accepts input from PSDabel and PSD Configuration, synthesizes this user logic and configuration, and fits the design to the PSD silicon. The Address Translator process allows the user to map the MCU firmware from a cross-compiler (in Intel HEX or S-Record format) into the NVM memory blocks within the PSD. As a result, the MCU firmware is merged with the logic and configuration definition of the PSD. The combined output of the Fitter and Address Translator is the .obj file that can be used by a programmer to program the PSD device. This .obj file can also be used to program a PSD813F using the JTAG FlashLink cable. The .obj file is comprised of the configuration, PLD fusemap, and MCU firmware.

3.4 PSD Simulator

WSI's version of SIMUCAD's SILOSIII simulation software provides functional chip-level simulation for PSD devices. PSDsoft automatically creates files for input to the simulator. These files convey relevant design information to the simulator. As a result, the user only has to create a stimulus file since all of the signals and node names are taken from the .abl file.

3.5 Parallel Programmer

Interface to the PSDpro programming device. It accepts the .obj file as input, allows viewing and editing of the .obj file, and programs and erases the PSD device.

3.6 JTAG Programmer

Interface to the FlashLink cable. It accepts the .obj file as input, and allows the PSD device to be programmed and erased in-system via the JTAG compatible FlashLink cable.

3.7 C code Generation

This is a new feature of PSDsoft that automatically generates C code functions and headers for controlling Flash PSD devices. These functions and headers are ANSI-C compatible. The generated files (.c and .h) may be edited to suit the particular application, then compiled and linked with the rest of the code. Afterwards, the linker output of the cross-compiler (usually in Intel HEX or Motorola S-record format) is merged with the configuration file of the PSD device in the Address Translate utility of PSDsoft.

The functions and headers provided by PSDsoft will cover key PSD operations such as: Flash memory program and erase algorithms, EEPROM program algorithms, I/O control and definition, memory management, and power management.

4 Design Flow

This section describes the design flow of a project—from entering the design in PSDabel—to programming the device and simulation. Figure 3 (right) shows the PSDsoft Design Flow utility. This is the first window to appear after you invoke PSDsoft. By double clicking on each box, the associated process is initiated. While this is a convenient method to navigate through the steps, this tutorial shows how to step through the process using menus and toolbars since this approach is less obvious. Section 5 takes you step-by-step through a tutorial design.

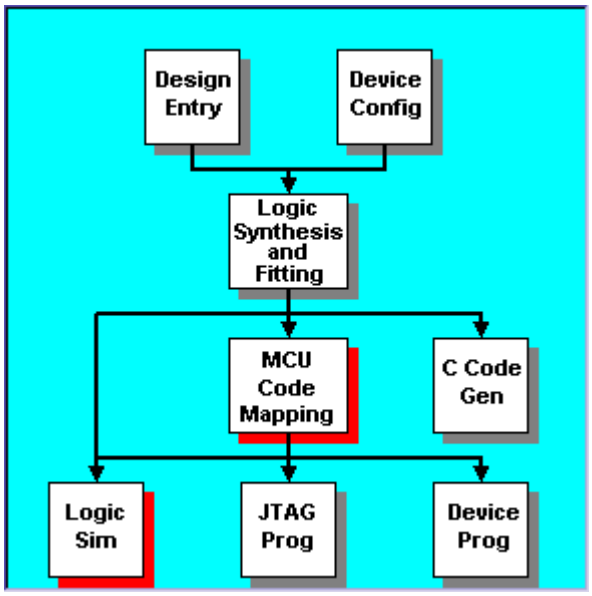


Figure 3—PSDsoft Design Flow

4.1 PSDsoft Program Flow

Here are the high-level steps to complete a PSD design.

1. Create or open a project after entry into PSDsoft. If you are creating a new project, specify the project name, the directory path, device family, part number, and provide a small description of the design if desired.

2. Select a design template (*project*.abl file), and modify this template to fit your design.

3. Use PSDabel to edit, compile, and optimize the *project*.abl file. Perform ABEL simulation if desired. To do so, you'll need to create necessary test vectors and place them at the end of the PSDabel file. A successful PSDabel compile operation will generate an optimized PLA file (*project*.tt2) for the Fitter.

4. Configure the PSD device using PSD Configuration; this generates the *project*.glc file for the Fitter.

5. Fit the design using PSD Fitter. The Fitter's input files are obtained from PSDabel and PSD Configuration. The Fitter generates the *project*.fob file that is passed on to the Address Translator. The Fitter also generates two fusemap files, *project*.afu and *project*.pfu for the Simulator. After a successful fit, it is possible to skip to step 8 (simulation) if desired since PSDsilosIII can be used before or after MCU firmware is merged with the PSD configuration.

6. Generate C code if desired. Edit this C code to suit your particular application. Then, compile and link it with your other application C code. Your cross-compiler will output an Intel HEX or Motorola S-record file containing the firmware.

7. Perform the address translation. The Address Translator combines the MCU firmware file and the *project*.fob file into a *project*.obj file. This *project*.obj file includes the MCU firmware, the fusemap, and the configuration bits.

8. Verify the design using PSD Simulator. Chip level simulation is based on the user's Verilog stimulus file (*project*.stl) and fusemap files from the Fitter. You must create the *project*.stl file. However, PSDsoft creates files to be used with the simulator that allow you to use the same names that appear in your *project*.abl file, and various reserved names.

9. Use PSDsoft to download the *project*.obj file to the PSD813F via a PSDpro or FlashLink JTAG programmer. A compatible third party programmer can also be used. Contact WSI or a representative near you for a list of compatible programmers.

5 813F Tutorial Example

This section uses the tutorial design example to illustrate the steps to implement the functionality discussed in section 3. The files required, which were generated for the tutorial design, can be found in the \PSDSOFT\TUTORIAL\TUTOR8XX\TUTOR directory.

At this point, you may wish to start the PSDsoft program, so that you can follow along with the tutorial example.

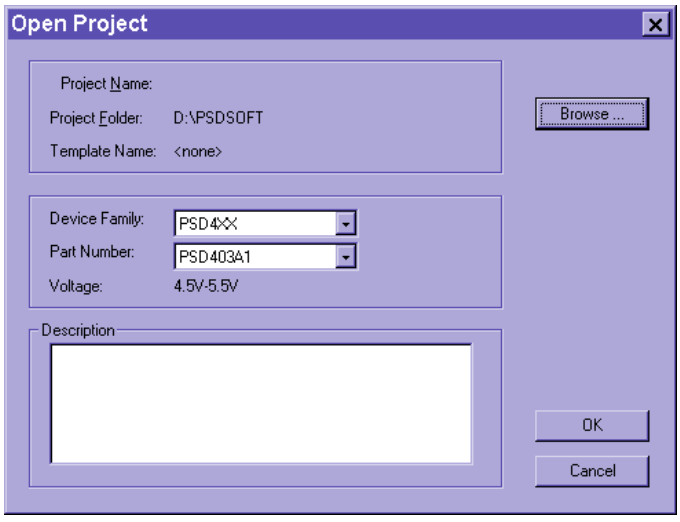
5.1 Managing the Project

Each new project may have its own working directory where all the files generated by PSDsoft reside. Once you specify the new project name, PSDsoft passes the working directory and pertinent information to other functional modules. In the following sections, you will be guided through a full sample design process, and key windows are displayed to help you follow the example.

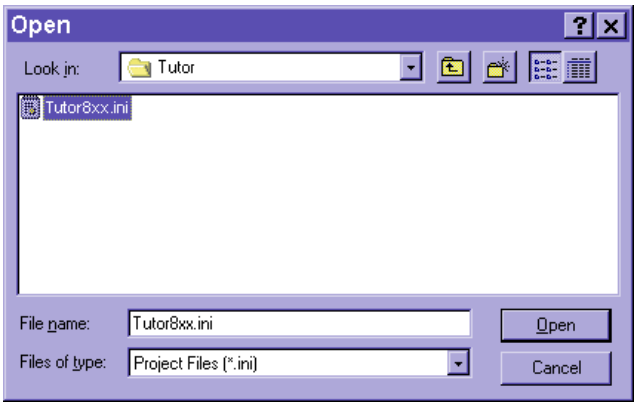
1. Start PSDsoft; a dialog box titled "PSDsoft" pops up, inquiring if you want to open and existing project or create a new one. Select "Open an existing project", and click **OK**.

Note, if you exited PSDsoft without closing whatever project you might have been working on, that project will automatically be reopened. If you don't see the above dialog box, pull down the **Project** menu and select **Open Project**.

2. The "Open Project" dialog box appears. Click the **Browse…** button.



3. The "Open" window appears. Go to the \PSDSOFT\TUTORIAL\TUTOR8XX\TUTOR directory, select the *tutor8xx.ini* file, and click on the **Open** button, which closes the "Open" dialog box.

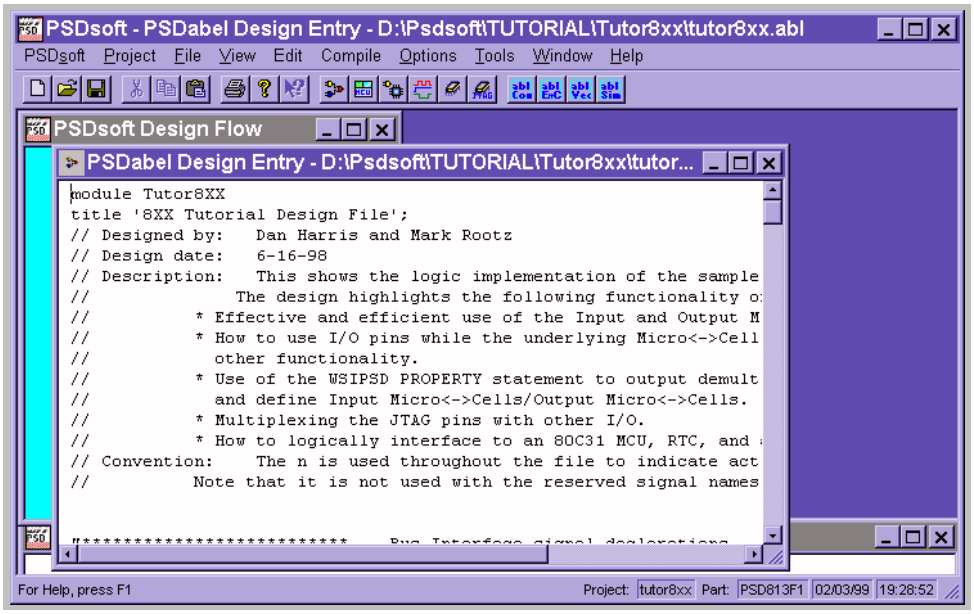


4. Click on the **OK** button, which closes the "Open Project" dialog box.

5.2 The PSDabel File

For detailed information on PSDabel and how it relates to the PSD813F, please read the comments in the file *tutor8xx.abl* in Appendix A. Also, refer to WSI's *Application Note 55* (*PSD GPLD Primer-PSD6XX/7XX/8XX*) and the *PSDsoft PSDabel-HDL Reference Manual*. For more information on the system memory map for this tutorial design, see Appendix F. To open the *tutor8xx.abl* design file, click on **View->Design File**, click the "Design Entry" button on the tool bar, or click on "Design Entry" in the design flow window.

Your design file should now be open, as shown below.

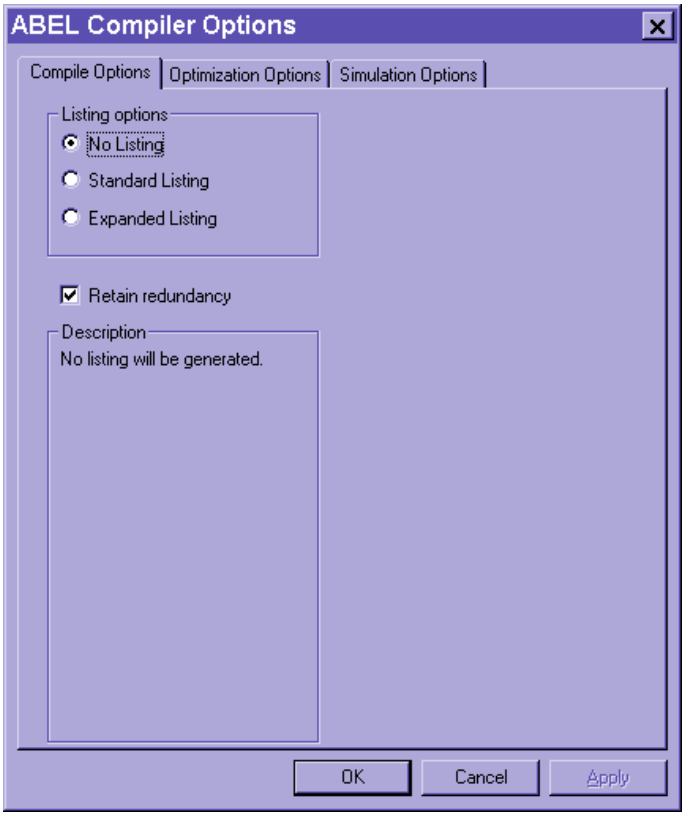

PSDsoft - PSDabel Design Entry - D:\Psdsoft\TUTORIAL\Tutor8xx\tutor8xx.abl
PSDsoft Project File View Edit Compile Options Tools Window Help
PSDsoft Design Flow
PSDabel Design Entry - D:\Psdsoft\TUTORIAL\Tutor8xx\tutor...
module Tutor8XX
title '8XX Tutorial Design File';
// Designed by: Dan Harris and Mark Rootz
// Design date: 6-16-98
// Description: This shows the logic implementation of the sample
// The design highlights the following functionality o
// * Effective and efficient use of the Input and Output M
// * How to use I/O pins while the underlying Micro<->Cell
// other functionality.
// * Use of the WSIPSD PROPERTY statement to output demult
// and define Input Micro<->Cells/Output Micro<->Cells.
// * Multiplexing the JTAG pins with other I/O.
// * How to logically interface to an 80C31 MCU, RTC, and
// Convention: The n is used throughout the file to indicate act
// Note that it is not used with the reserved signal names
For Help, press F1
Project: tutor8xx Part: PSD813F1 02/03/99 19:28:52

5.2.1 Compiling the Tutor Design

To compile the *tutor8xx.abl* file, take the following steps:

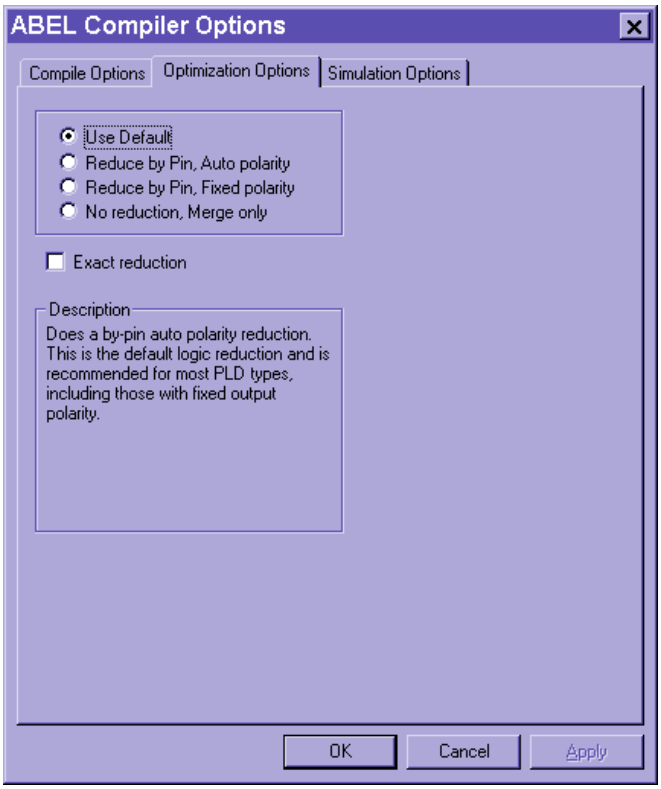
1. Click on the **Options** menu and select **ABEL Compiler Options…**.

2. The "ABEL Compiler Options" dialog box appears. Click on each of the options, and read the description in the "Description" box to get a feel for what each option will do. Then set up the options as shown below, with the **Standard Listing** selected under "Listing options" and the **Retain redundancy** box checked. For a better description of the various options available, refer to the *PSDabel—HDL Reference Manual*.

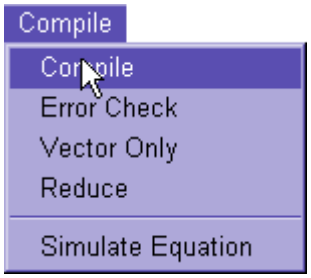


3. Select the “Optimization Options” tab, and set up the options as shown below. **Use Default** should be the only item selected.

4. Click on the **OK** button when you have finished setting up the options.


ABEL Compiler Options
Compile Options
Optimization Options
Simulation Options
Use Default
Reduce by Pin, Auto polarity
Reduce by Pin, Fixed polarity
No reduction, Merge only
Exact reduction
Description
Does a by-pin auto polarity reduction. This is the default logic reduction and is recommended for most PLD types, including those with fixed output polarity.
OK
Cancel
Apply

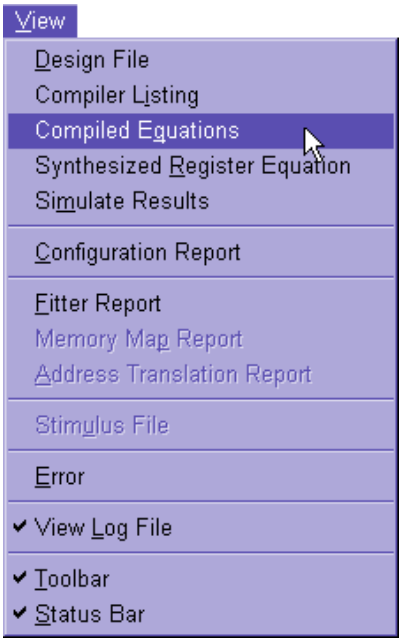
5. Click on **Compile->Compile**.



Or, click on the “abl Com” button on the tool bar.

6. The PSDabel compiler generates an error file—*tutor8xx.err* (even if no errors are present), and writes to the log file—*tutor8xx.plg*. The compiler also generates a PLA output file, *tutor8xx.tt2*, which is used by PSDsoft for fitting, and is optimized based on the reduction algorithm specified in the “Optimization Options” under the **Options** menu.

7. After compilation, you can display the optimized PLD logic equations that will be used by the Fitter by pulling down the **VIEW** menu and select the **Compiled Equations**; this opens the *tutor8xx.eq2* file.



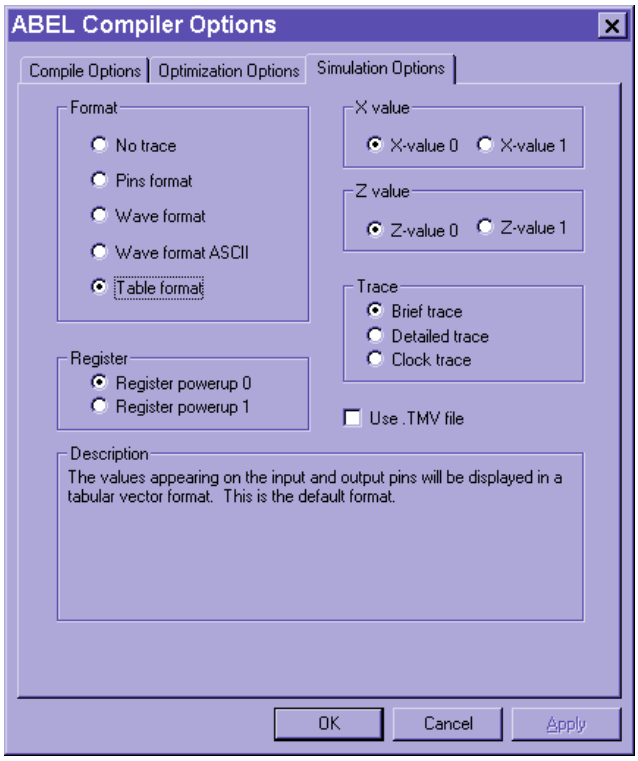
5.2.2 Simulating your design using ABEL simulation

You can do a very simplistic functional simulation of the blocks that make up the PLD using the simulator that is included with PSDabel. It is important to note that only the functions that are generated within the .abl file can be simulated using test vectors at the end of the file. For chip-level functional simulation, you must have the version of PSDsoft that includes the PSDsilosIII simulation software. To use the simulator that comes with PSDabel, take the following steps:

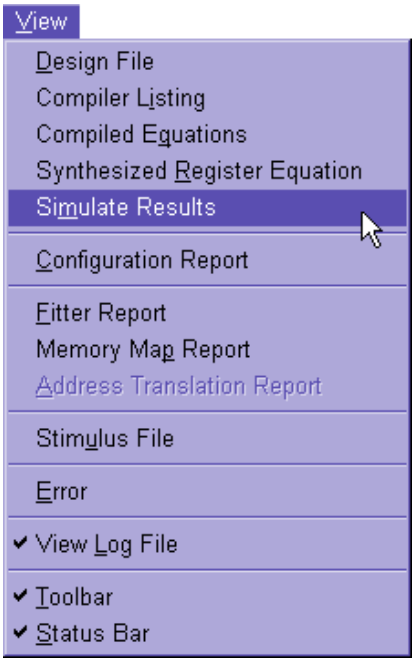
1. Click on the **Options** menu and select **ABEL Compiler Options…**.

2. Once the "ABEL Compiler Options" dialog box appears, click on the "Simulator Options" tab, and set up the window as shown: under "Format", choose **Table format**. Ensure that the **X-value 0** and **Z-value 0** are selected in their respective boxes, and that **Brief trace** is selected in the "Trace" box. For the "Register" box, select the **Register powerup 0**, and make sure that the "Use .TMV file" box is not checked.

3. Click on the **OK** button to save your changes.

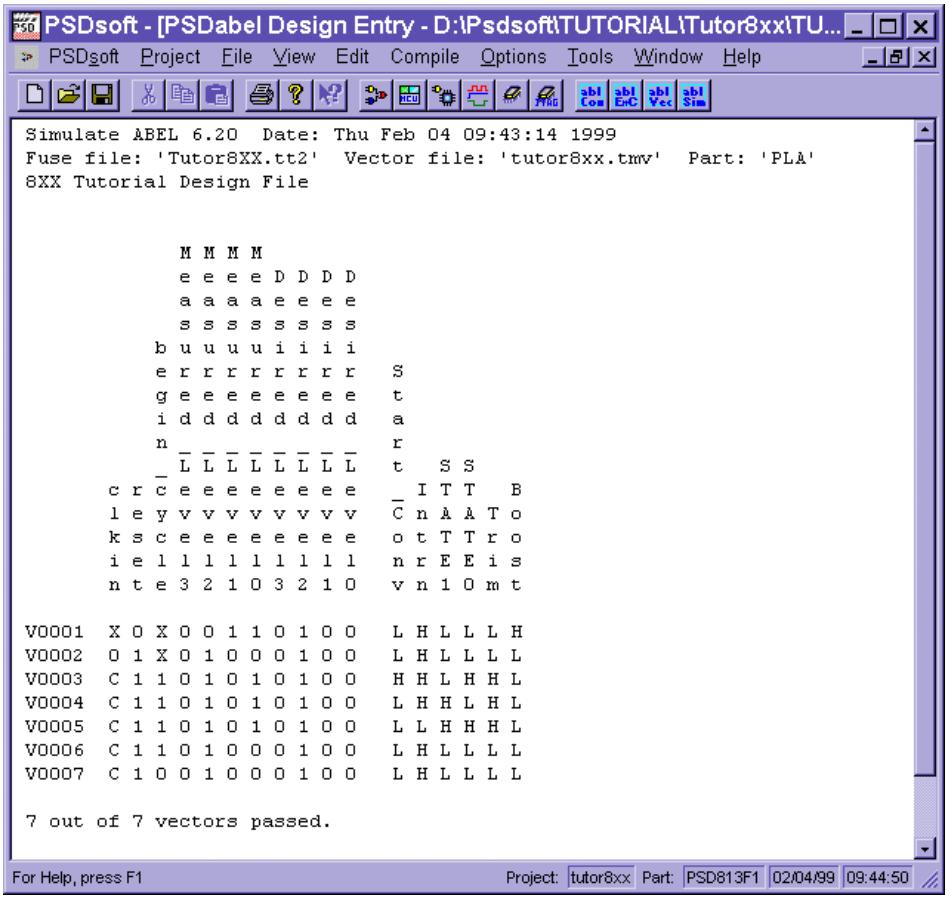


4. If you select **Simulate Results** in the **View** menu, PSDabel will automatically start the simulation process, and display the simulation results based on the logic equations and test vectors in the .abl file.



> Note: the ABEL file has four lines that should be commented out for the ABEL simulation only. If these lines are not commented out, you will get a message that 0 of 7 vectors passed in the ABEL simulation instead of the display below. The lines that need to be commented out are highlighted in the "CPLD equations" section. If you decide to edit the ABEL file, and wish to run the PSDSilosIII simulation later, you must un-comment the lines.

Your simulation results should look like the screen capture below. (See above note.)

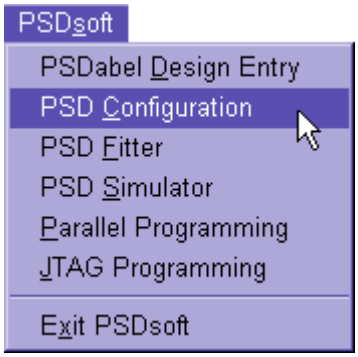

PSDsoft - [PSDabel Design Entry - D:\Psdsoft\TUTORIAL\Tutor8xx\TU...
PSDsoft Project File View Edit Compile Options Tools Window Help
Simulate ABEL 6.20 Date: Thu Feb 04 09:43:14 1999
Fuse file: 'Tutor8XX.tt2' Vector file: 'tutor8xx.tmv' Part: 'PLA'
8XX Tutorial Design File
M M M M
e e e e D D D D
a a a a e e e e
s s s s s s s s
b u u u u i i i i
e r r r r r r r r S
g e e e e e e e e t
i d d d d d d d d a
n _ _ _ _ _ _ _ _ r
_ L L L L L L L L t S S
c r c e e e e e e e e _ I T T B
l e y v v v v v v v v C n A A T o
k s c e e e e e e e e o t T T r o
i e l l l l l l l l n r E E i s
n t e 3 2 1 0 3 2 1 0 v n 1 0 m t
V0001 X 0 X 0 0 1 1 0 1 0 0 L H L L L H
V0002 0 1 X 0 1 0 0 0 1 0 0 L H L L L L
V0003 C 1 1 0 1 0 1 0 1 0 0 H H L H H L
V0004 C 1 1 0 1 0 1 0 1 0 0 L H H L H L
V0005 C 1 1 0 1 0 1 0 1 0 0 L L H H H L
V0006 C 1 1 0 1 0 0 0 1 0 0 L H L L L L
V0007 C 1 0 0 1 0 0 0 1 0 0 L H L L L L
7 out of 7 vectors passed.
For Help, press F1
Project: tutor8xx Part: PSD813F1 02/04/99 09:44:50

5.3 PSDsoft Configuration

The PSD813F has a programmable MCU bus interface and is able to interface directly to many microcontrollers. Using PSD Configuration, you can specify how to interface to the MCU you have chosen for your design. You can also configure functions specific to the PSD device you are using. This tutorial design is based on the Intel 80C31 microcontroller, which has an 8-bit multiplexed bus with /RD, /WR, and /PSEN as the control signals, and an active-high level address latch enable (ALE).

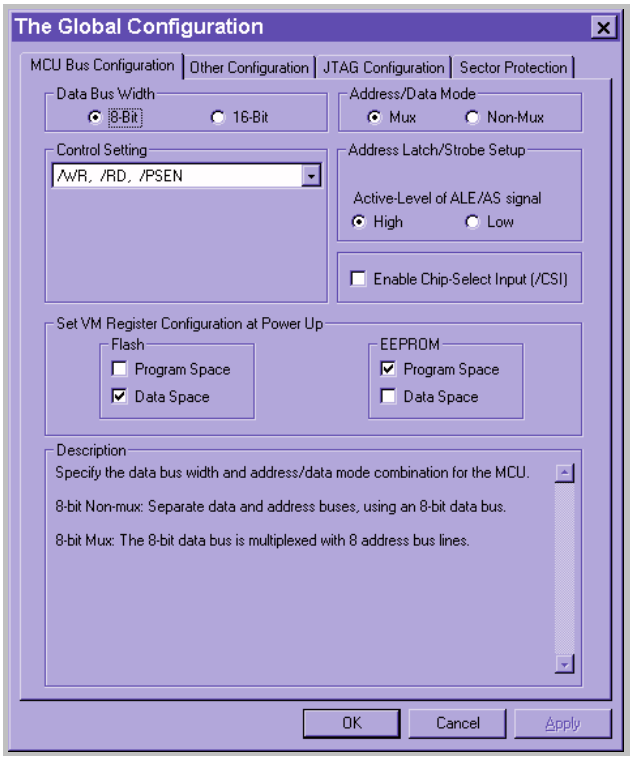
To perform the configuration, take the following steps:

1. Pull down the **PSDsoft** menu in the main PSDsoft window and choose **PSD Configuration**, click the "Configuration" button, or click on "Device Config" in the "PSDsoft Design Flow" window.

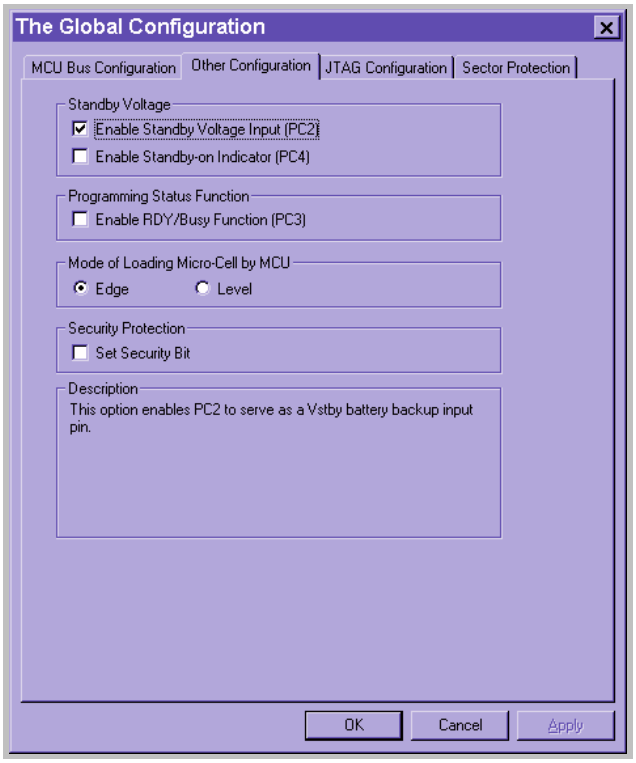


2. A dialog box opens titled "PSD Configuration". Make sure the "MCU Bus Configuration" tab is selected. Set up the configuration as shown:
   - Ensure **8-Bit** is selected under "Data Bus Width"
   - **Mux** is selected under "Address/Data Mode"
   - **High** is selected under "Address Latch/Strobe Setup"
   - The "Enable Chip-Select Input (/CSI)" box is not checked
   - The **/WR, /RD, /PSEN** is selected under "Control Setting"
   - **Data Space** is selected under "Flash"
   - **Program Space** is checked under "EEPROM".

   This arrangement for program and data space allows the MCU to boot from EEPROM in program space and download to Flash memory in data space if needed. Afterwards, the MCU can override this arrangement if, for example, it wanted to move Flash memory to program space. This can be done by the MCU writing the VM register.

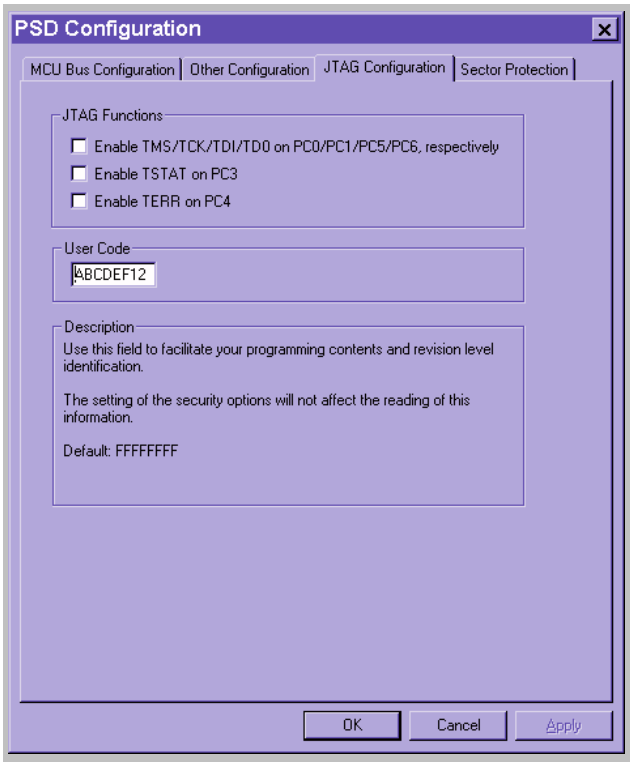


3. Click on the “Other Configuration” tab, and ensure that the **Enable Standby Voltage Input (PC2)** box is checked under “Standby Voltage”, **Edge** is selected under “Mode of Loading Micro⇔Cell by MCU”, and all other boxes are unchecked.



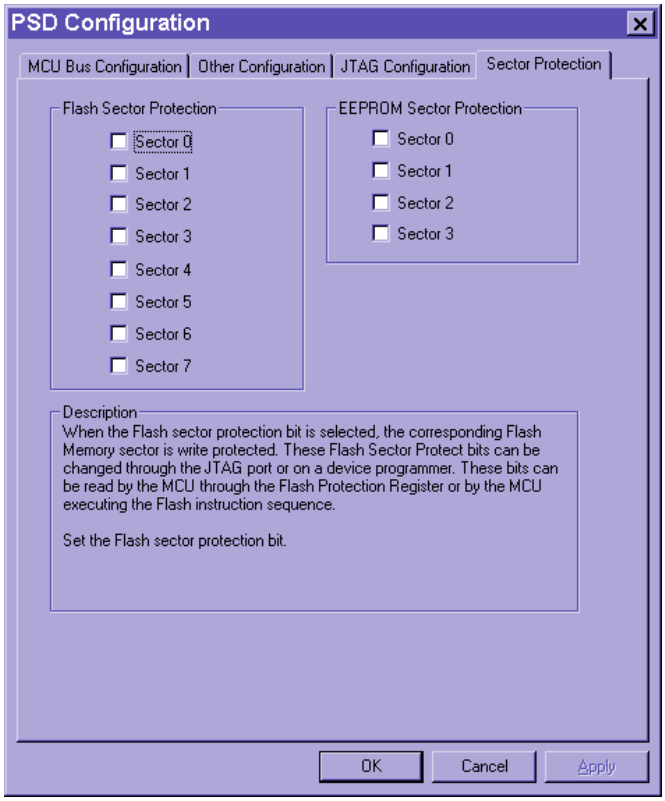
4. Click on the “JTAG Configuration” tab and ensure that none of the boxes are checked because checking the boxes would enable the JTAG port to be operational 100% of the time. Since in this tutorial we are multiplexing the JTAG pins with other signal functions, it is desired that JTAG functions only be operational when the JEN/ signal is active (see the Figure 2 schematic).

   Enter the value ‘ABCDEF12’ in the “User Code” box below. This value will be programmed into your PSD device. The User Code can be any value you wish (e.g. to identify end product software revisions, serial numbers, etc.). Up to eight hex characters may be entered.

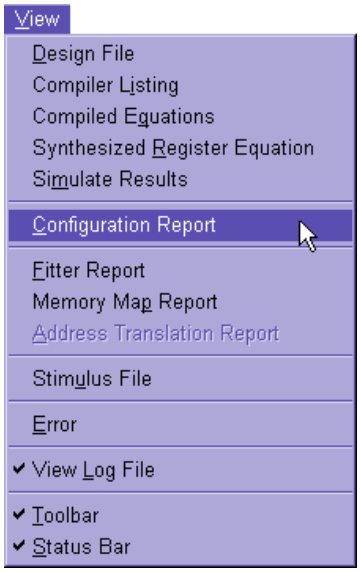


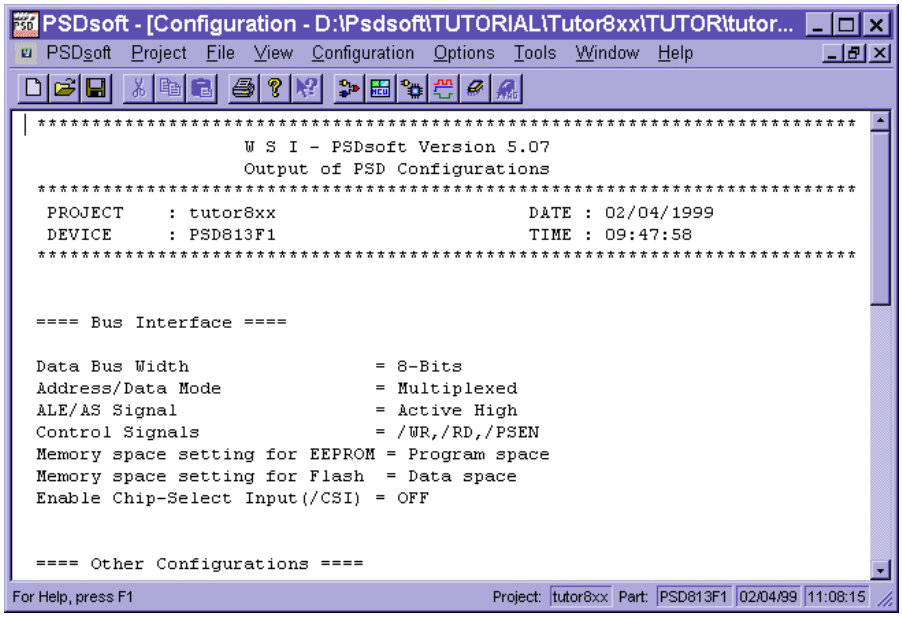
5. Click on the “Sector Protection” tab, and ensure that none of the boxes are checked. The appropriate sector box should only be checked if it is desired that the selected sector be write protected. These bits can be changed later through the JTAG port or the device programmer.

6. When you are finished with the configuration settings, click on the **OK** button, which saves the configuration. The PSD813F configuration is now completed.


PSD Configuration
MCU Bus Configuration
Other Configuration
JTAG Configuration
Sector Protection
Flash Sector Protection
Sector 0
Sector 1
Sector 2
Sector 3
Sector 4
Sector 5
Sector 6
Sector 7
EEPROM Sector Protection
Sector 0
Sector 1
Sector 2
Sector 3
Description
When the Flash sector protection bit is selected, the corresponding Flash Memory sector is write protected. These Flash Sector Protect bits can be changed through the JTAG port or on a device programmer. These bits can be read by the MCU through the Flash Protection Register or by the MCU executing the Flash instruction sequence.
Set the Flash sector protection bit.
OK
Cancel
Apply

If you ever wish to view the configuration file, first ensure you are in Configuration Mode. (See Step 1 of this section.) Next, pull down the **View** menu and select **Configuration Report**. The configuration report is shown below. To print the configuration report, select **File->Print**.




PSDsoft - [Configuration - D:\Psdsoft\TUTORIAL\Tutor8xx\TUTOR\tutor...
PSDsoft Project File View Configuration Options Tools Window Help
W S I - PSDsoft Version 5.07
Output of PSD Configurations
PROJECT : tutor8xx
DATE : 02/04/1999
DEVICE : PSD813F1
TIME : 09:47:58
==== Bus Interface ====
Data Bus Width = 8-Bits
Address/Data Mode = Multiplexed
ALE/AS Signal = Active High
Control Signals = /WR,/RD,/PSEN
Memory space setting for EEPROM = Program space
Memory space setting for Flash = Data space
Enable Chip-Select Input(/CSI) = OFF
==== Other Configurations ====
For Help, press F1
Project: tutor8xx
Part: PSD813F1
02/04/99
11:08:15

5.4 PSD Fitter : Fitting and Address Translation

PSD Fitter consists of the Fitter and the Address Translator. The Fitter accepts input from PSDabel and PSD Configuration, synthesizes the user logic and configuration, and fits the design to the PSD813F silicon. The Address Translator process allows the user to map the MCU firmware from a cross-compiler (in Intel HEX or S-Record format) into the NVM memory blocks within the PSD. As a result, the MCU firmware is merged with the logic and configuration definition of the PSD. The combined output of the Fitter and Address Translator is the *tutor8xx.obj* file.

5.4.1 Fitting the Design

The input files to the Fitter are:

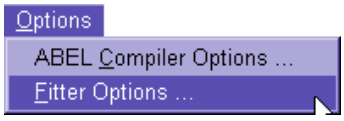
tutor8xx.tt2—PLA file generated by PSDabel.
tutor8xx.glc—PSD813F configuration file generated by PSD Configuration.

The output files generated by the Fitter are :

tutor8xx.fob—PLD fusemap and PSD813F configuration file.
tutor8xx.afu—Generated for use by PSDSilosIII.
tutor8xx.pfu—Generated for use by PSDSilosIII.
tutor8xx.obj—Object file (PLD and Configuration portion only).
tutor8xx.frp—Fitter report file.

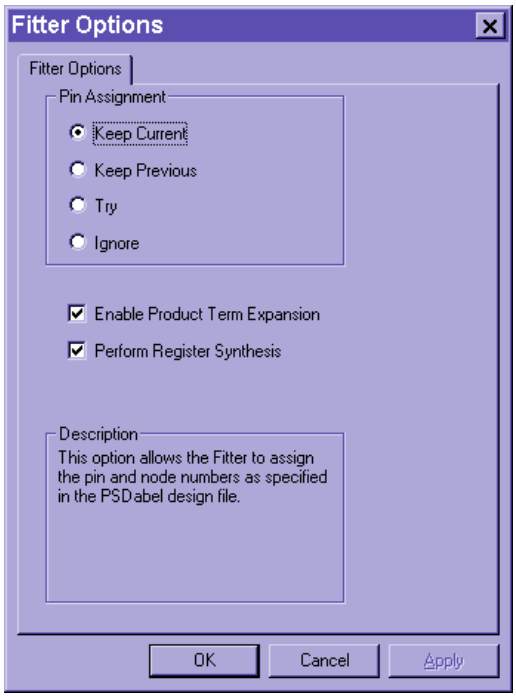
To Fit a Design

1. Click on the **Options** Menu, and select the "Fitter Options."



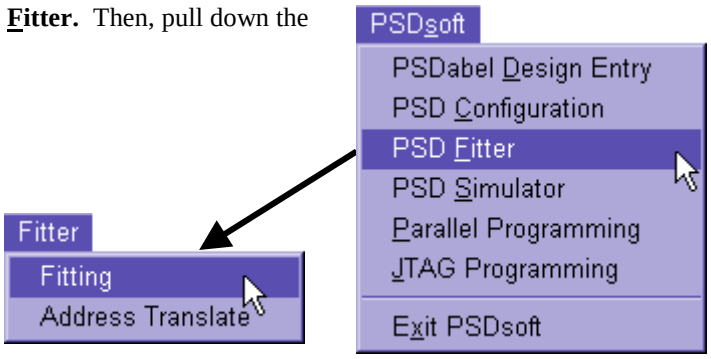
2. For the tutorial, choose **Keep Current** under "Pin Assignment", and ensure that the "Enable Product Term Expansion" and "Perform Register Synthesis" boxes are checked, as shown.

3. Click **OK** to save the Fitter options.



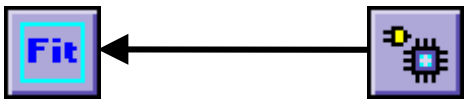
Do This

4. Pull down the **PSDsoft** menu and choose **PSD Fitter.** Then, pull down the **Fitter** menu and choose **Fitting.**



OR This

Or, click the "Fitter" and then the "Fit" button on the tool bar.



Or This

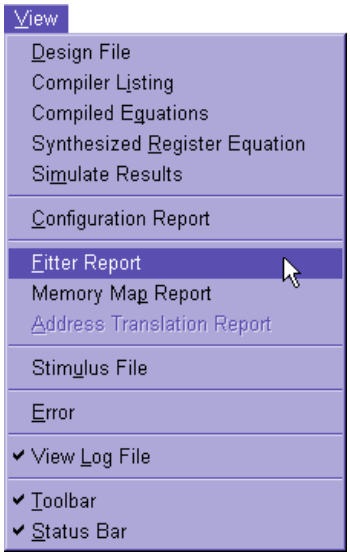
Or, click the "Logic Synthesis and Fitting" box in the design flow and the Fitter automatically runs.

5. The Fitter appends to two files: the log file (*PSDsoft.plg*) and the error file (*tutor8xx.err*). Check the log file for any possible errors. If there are no errors present (there shouldn't be if you didn't modify the *tutor8xx.abl* file), skip to Step 8.

6. If the fitting is not successful, you may have to view the *tutor8xx.eq2* file in PSDabel to see which logic function caused the fitting problem and modify the *tutor8xx.abl* file accordingly. To view the optimized equation file (*tutor8xx.eq2*), see step 7 in the "Compiling the Tutor Design" section (5.2.1).

7. Recompile the modified *tutor8xx.abl* file. Repeat Steps 4 through 7 until a successful fit has been found. Then, re-enter the Fitter program, and proceed to Step 8.

8. Examine the Fitter Report File by pulling down the **VIEW** menu and selecting **Fitter Report**. The report file shows the results of the fitting process, and the pin assignment for the PSD813F1. If you want a certain fitting other than the one generated, return to the *tutor8xx.abl* file to change the desired signal and pin assignments.



You may also view the Memory Map Report (*tutor8xx.map*) by selecting **View->Memory Map Report.** It contains information on how the DPLD will interpret addresses from the MCU, and shows the memory map for internal memory blocks and relevant external signals. This information is based entirly on the design (.abl) file and is summarized for your convenience.

Before we cover the Address Translator, you should be aware of a new PSDsoft feature unique to Flash PSD devices—automatic C code generation.

5.4.2 Generating C code

PSDsoft can generate ANSI C code functions and headers for controlling the PSD813F. This is an optional step; however, it will save you time by implementing low-level PSD driver function and header files. The functions and headers are ANSI-C compatible. The .c and .h files that are generated should be edited to suit your application, then compiled and linked with the rest of your application code using an MCU cross-compiler and linker.

The functions and headers that can be generated by PSDsoft include the following operations:

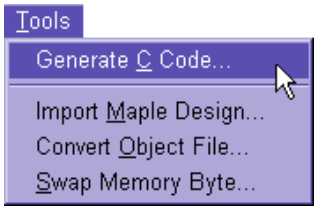
- ✓ Flash memory program and erase algorithms
- ✓ EEPROM program algorithms
- ✓ I/O control and definition
- ✓ Memory and power management
- ✓ Others coming soon….

Although C code generation can be performed anytime after a project is opened, we recommend it be done after you've successfully fit your design. Once a successful fit is achieved, all pin functions and PSD configurations are defined, and the C code may be tailored accordingly.

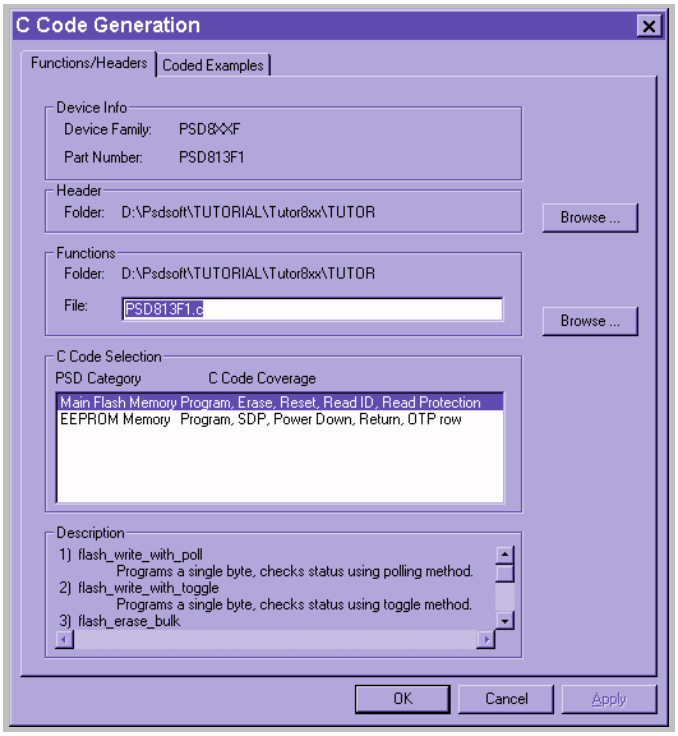
The source C programming files to implement the AGC function for this tutorial have not been provided; however, since this tutorial is meant to cover all aspects of a PSD813F design, we cover how you would use the C code generation utility for your own project.

Take the following steps to generate C code:

1. Pull down the **Tools** menu in the PSDsoft window and choose **Generate C Code…**, as shown. Alternately, click the "C Code Gen" button in the Design Flow window.



2. The “C Code Generation” dialog box should appear:



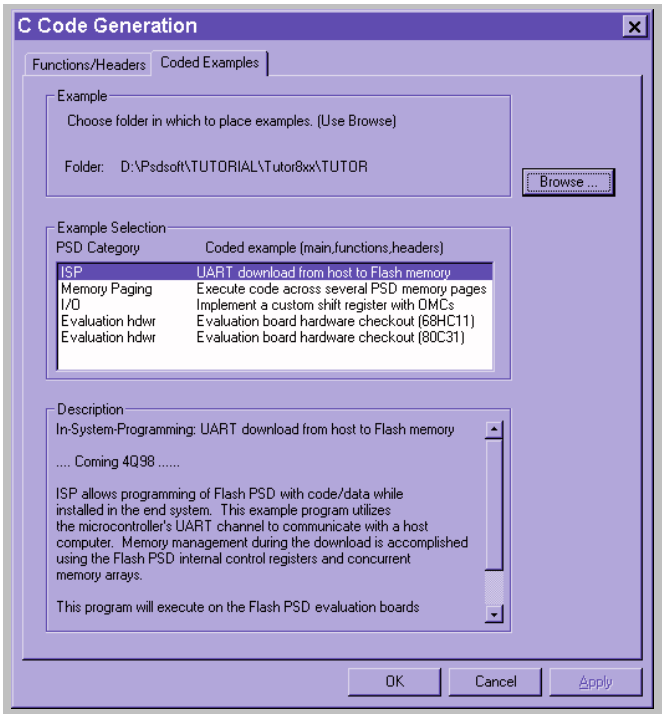
The “Functions/Headers” dialog box has the following sections:

- Device Info—PSD family and part number of the current project. These values cannot be changed unless this project is closed and a different one is opened.
- Header—Specify the folder that you would like to place the C header files (.h) generated by PSDsoft by clicking on the upper **Browse...** button. Typically, a folder in your MCU cross-compiler environment is chosen. You cannot change the name of the headers file(s) at this point since these header files may referenced by name within other header files or the C functions that are also generated by PSDsoft. Once all of the headers and functions are copied to their designated folders, you may edit the header file names any way you wish, as long as you change their names in the respective ‘#include’ statements.
- Functions—Specify the folder in which you would like to place the C function file (.c) by clicking the upper **Browse...** button. Typically, a folder in your MCU cross-compiler environment is chosen. This file will contain the functions you specify in the next section.
- C Code Selection—Select the categories of C code functions that you would like to integrate into your C application program. Under “PSD Category” are the major PSD functional groups that are supported with C code for the PSD device that is used for this project. Under “C Code Coverage” is a brief list of the individual functions that are available within each category. To select more than one category, hold the ‘Ctrl’ key while making selections with the left mouse button. **Note**: even if more than one category is selected, only one .c file is generated because functions are appended within the same file.
- Description—Offers a description of the functions that are generated if selected in the “C Code Selection” box. **Note:** if you double-click on a function within the Description box, the C code that will be generated is shown so you can get an idea of what will appear in the sample C file.

3. After you have made your selection, click **Apply** to generate the code. In this example, three files will be written to your folder(s), which are:
    - psd813F1.c—ANSI-C source for all of the selected functions
    - psd813F1.h—ANSI-C header file to define particular PSD registers
    - map813F1.h—ANSI-C header file to define locations of system memory elements (Flash, EEPROM, PSD registers, etc.).

Note: the file psd813F1.h contains define statements for each individual C function within the PSD813F1.c file. Later, edit psd813F1.h, and simply remove the comment delimiters (//) from the define statement for any C function that you would like to be compiled with the rest of your C source code.

4. Click on the "Coded Examples" tab at the top of the dialog box.


C Code Generation
Functions/Headers
Coded Examples
Example
Choose folder in which to place examples. (Use Browse)
Folder: D:\Psdsoft\TUTORIAL\Tutor8xx\TUTOR
Browse ...
Example Selection
PSD Category
Coded example (main,functions,headers)
ISP
UART download from host to Flash memory
Memory Paging
Execute code across several PSD memory pages
I/O
Implement a custom shift register with OMCs
Evaluation hdwr
Evaluation board hardware checkout (68HC11)
Evaluation hdwr
Evaluation board hardware checkout (80C31)
Description
In-System-Programming: UART download from host to Flash memory
.... Coming 4Q98 .....
ISP allows programming of Flash PSD with code/data while installed in the end system. This example program utilizes the microcontroller's UART channel to communicate with a host computer. Memory management during the download is accomplished using the Flash PSD internal control registers and concurrent memory arrays.
This program will execute on the Flash PSD evaluation boards
OK
Cancel
Apply

This sheet contains several examples that you may use as a basis for building your own C code application. These are complete projects (main, functions, and headers) targeted toward a particular MCU. You may copy these files to some folder to browse them for ideas, or cut and paste sections from the examples into your own cross-compiler environment. There are three sections:

- Example—Specify the folder that you would like to place the example project files generated by PSDsoft by clicking on the **Browse…** button and selecting a folder.
- Example Selection—There are several areas for which you may generate C code. Each category implements a high-level system function, such as memory paging or UART downloads to Flash memory.
- Description—Describes each of the coded examples in the "Example Section"

Once the C code generated by PSDsoft is integrated into your own C application and is successfully compiled and linked by your MCU cross-compiler, you are ready for address translation. Choose the desired example and click **Apply** for each example that you want generated. Click **OK** when you have finished making your selections.

5.4.3 Performing the Address Translation

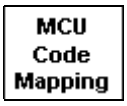
The Address Translator combines the *tutor8xx.fob* file with the MCU firmware file(s) generated by your choice of MCU cross-compilers. Address Translator will append to the *tutor8xx.obj* file that is to be downloaded to a programmer compatible with the PSD813F.

> **NOTE:** The addresses within in the generated *tutor8xx.obj* file are special "direct" addresses that are meaningful to a programming device. They are not "system" addresses that the MCU would use, or that the DPLD decodes. That is what is meant by 'Address Translate'. It is a translation of "system" addresses that the MCU and its compiler/linker know about to a set of "direct" addresses that a device programmer knows about.

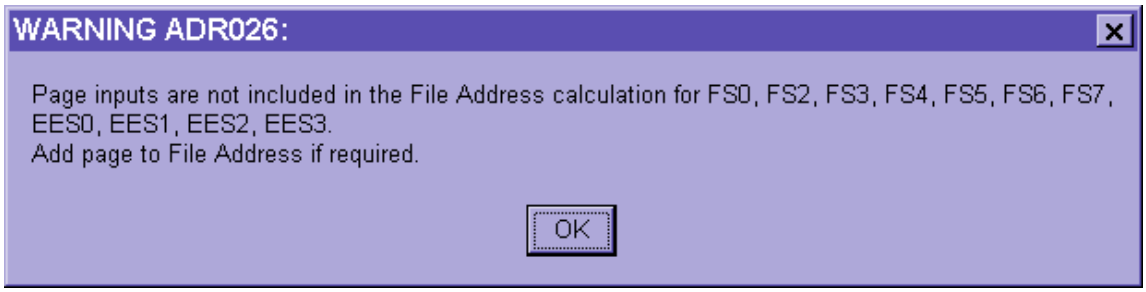
To perform the address translation, take the following steps:

1. Pull down the **PSDsoft** menu, and choose **PSD Fitter**. Then, pull down the **Fitter** menu and choose **Address Translate** or click the "Address Translate" button on the tool bar.

Or, select "MCU Code Mapping" in the design flow.

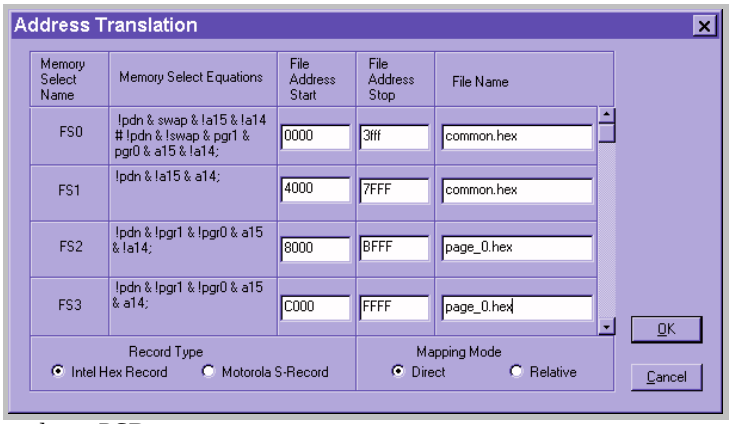


You will get the following message upon starting Address Translate:


WARNING ADR026:
Page inputs are not included in the File Address calculation for FS0, FS2, FS3, FS4, FS5, FS6, FS7, EES0, EES1, EES2, EES3.
Add page to File Address if required.
OK

This warning is a reminder to ensure that you take paging into account when entering the start/stop addresses and file names. Click **OK**.

2. The "Address Translate" dialog box appears. The Address Translation dialog box has the following sections:



- Memory Select Name—Name of the PSD memory segment that will be selected when the associated equation is true.
- Memory Select Equations—Each cell shows the equation for the appropriate PSD memory segment. These are the optimized equations from the PSDabel file. They are displayed for convenience, and cannot be modified in this window.
- File Address Start—Starting MCU system address from MCU compiler/linker that will be mapped to a PSD memory segment.
- File Address Stop—Ending MCU system address from MCU compiler/linker that will be mapped to a PSD memory segment.
- File Name—MCU firmware file that is generated by your MCU Compiler/Linker.
- Record Type—The supported formats are Intel HEX or Motorola S-Record.
- Mapping Mode—Two modes of mapping are supported, direct and relative. For more information, consult the *PSDsoft User Manual*.

Notice that PSDsoft attempted to fill in the File Start and File Stop Addresses based on your PSDabel equations. However, if paging is used (as in this tutorial), these file addresses must be

handled carefully since PSDsoft does not know how your MCU cross-compiler and linker handles paging. As we progress, this process should become clear.

3. Type in the file names of your MCU linker output in the appropriate places. In this example, five files are used. (See Appendix F for information on the system memory map and how these files relate.) Four of the five files are to be programmed into Flash memory on different pages. The remaining file is to be programmed into the boot area of the EEPROM. The four Flash files are *page_0.hex*, *page_1.hex*, *page_2.hex*, and *common.hex*. The file for the EEPROM is *boot.hex*. Each of these files can contain up to 32 Kbytes of code.

4. Enter the File Start Addresses, File Stop Addresses, and File Names according to the table below:

Memory Select	File Start Address	File Stop Address	File Name
FS0	0000	3FFF	Common.hex
FS1	4000	7FFF	Common.hex
FS2	8000	BFFF	Page_0.hex
FS3	C000	FFFF	Page_0.hex
FS4	8000	BFFF	Page_1.hex
FS5	C000	FFFF	Page_1.hex
FS6	8000	BFFF	Page_2.hex
FS7	C000	FFFF	Page_2.hex
EES0	0000	1FFF	boot.hex
EES1	2000	3FFF	boot.hex
EES2	-	-	-
EES3	-	-	-

In this design, a different file name was used for each of several sections of code in Flash memory because of the address space overlap of the segments. This file scheme is used because, even though these sections of code physically reside on different memory pages, some linkers will place them in overlapping absolute address space. The method you use depends on your linker. Alternatively, you can use a single file name across many memory chip selects if your linker automatically appends extra address bits that represent your paging scheme. You would then, for example, enter 18-bit addresses to accompany the single file name (which is passed to the Address Translate utility), instead of 16-bit addresses to accompany several file names.

NOTE: Optionally, you can specify only the EEPROM contents to be programmed by the device programmer. It may be desired to load system code into Flash memory while in-system, not on a device programmer. In this case, only information for EES0 and EES1 should be entered in the Address Translate utility.

5. Ensure that **Direct Mapping** is selected in the "Mapping Mode" box.

6. Select **Intel Hex Record** in the "Record Type" box.

7. Click on **OK** to perform the address translation. If no errors are indicated, then *tutor8xx.obj* will be appended.

If your copy of PSDsoft includes the PSDsilosIII simulator, you should simulate and verify your design before programming the PSD813F. Refer to the next section on how to simulate the tutorial design.

5.5 PSD813F Chip Simulation

PSDsilosIII is WSI's version of SIMUCAD's SILOSIII simulator software. It provides chip-level simulation and design verification using the Verilog Hardware Description Language (Verilog-HDL). Appendix B lists the stimulus file (*tutor8xx.stl*) for this tutorial.

Many of the internal nodes on the PSD813F are available for tracing. Descriptions of the signals that can be traced by the simulator are listed in Appendix C.

PSDsoft generates all but one of the input files required by the simulator. The file that must be created is the stimulus file (.stl). In the stimulus file, you can use the same names you used in your PSDabel file, and the predefined ones in Appendix C.

5.5.1 *PSDsoft.run* File

One of the files generated by PSDsoft for the simulation process is *PSDsoft.run*, listed in Figure 4 below. It is a command batch file used by PSDsilosIII. For additional information on PSDsilosIII commands (commands starting with !), refer to PSDsilosIII's on-line help.

Figure 4 *PSDsoft.run* File

```
!Reset all
!file .sav = Tutor8xx
!control .ext = all
`timescale 1ns/0.1ns
!lib d:\psdsoft\psd8.v
`include "tutor8xx.top"
`include "tutor8xx.stl"
endmodule
```

Let's analyze the *PSDsoft.run* file:

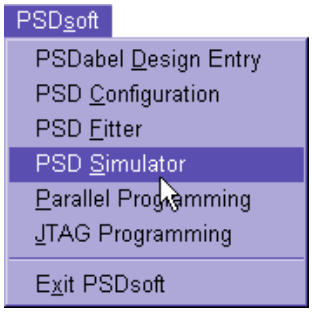
- !Reset all—tells the compiler to reset all the input signals to their default state.
- !file .sav = Tutor8xx—tells the compiler the project name is "Tutor8xx."
- `timescale 1ns/0.1ns—compiler directive for defining the delay values for a modul. 1 ns is the unit of measurement for times and delays and 0.1 ns is the precision to which the delays are rounded off.
- !lib d:\psdsoft\psd8.v—specifies the library file to be used. There is one important thing to note about the included library file: it looks for other files automatically generated by PSDsoft from the fusemap file that have a .afu and .pfu extension. They allow simulation of the logic defined in the .abl file in the stimulus file.
- `include—compiler directive that allows the entire contents of a Verilog source file to be included in another file (*PSDsoft.run* in this case). *Tutor8xx.top* is generated by PSDsoft based on the PSDabel file, and allows you to use of any of the signal names within the PSDabel file. There are also parameter definitions for high impedance state signals (Z1 through Z32) in the .top file. The *tutor8xx.stl* file is the user-created stimulus file. (See Appendix B.)
- The "endmodule" statement is the last statement in the *PSDsoft.run* file. It is there because it complements the "module WSIdesign" statement in the .top file.

5.5.2 Running the Logic Simulator

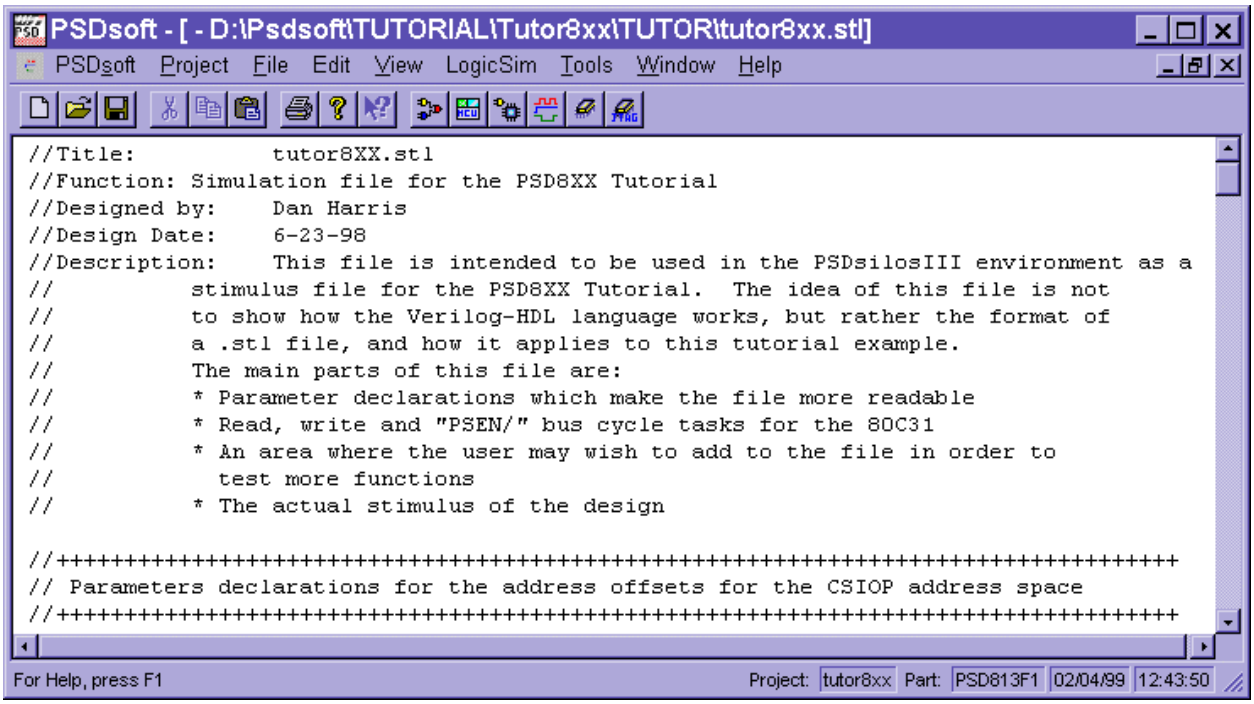
To run the simulator, take the following steps:

1. Review the stimulus file (*tutor8xx.stl*) listed in Appendix B.

2. Pull down the **PSDsoft** menu in the main PSDsoft window and select **PSD Simulator** or click the "simulator" button on the tool bar. This open up the *tutor8xx.stl* stimulus file for editing within PSDsoft.



3. The *tutor8xx.stl* file is automatically opened in PSDsoft, as shown.

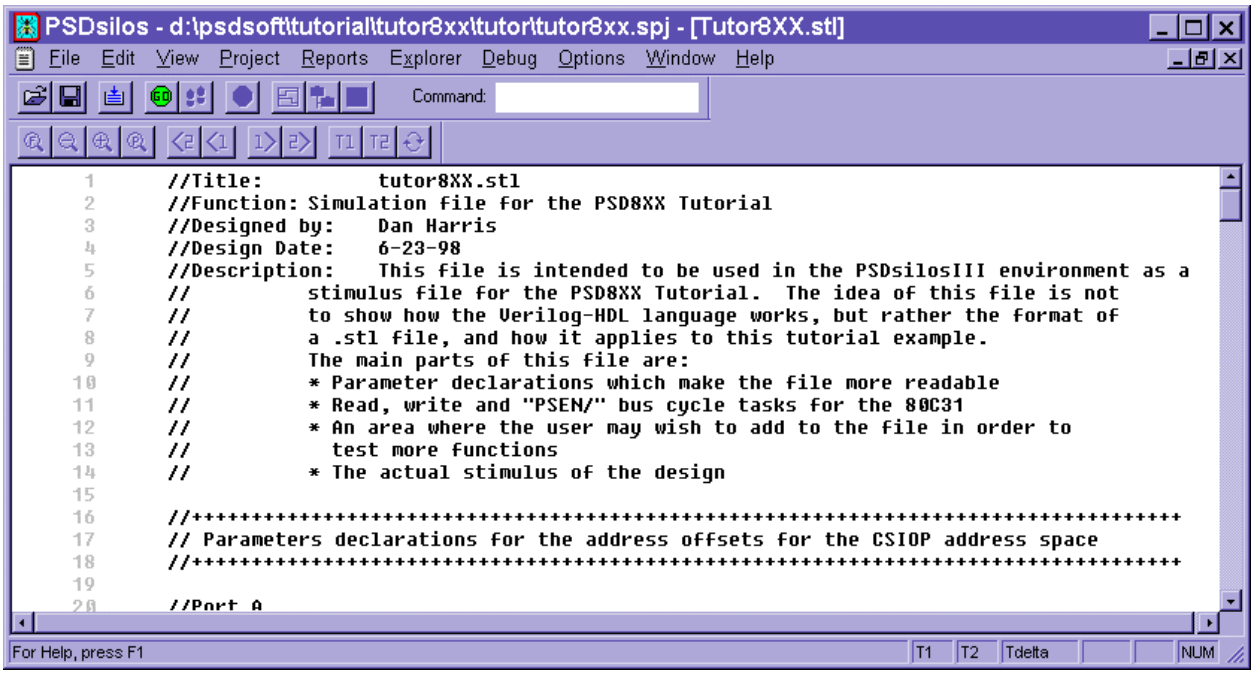

PSDsoft - [- D:\Psdsoft\TUTORIAL\Tutor8xx\TUTOR\tutor8xx.stl]
PSDsoft Project File Edit View LogicSim Tools Window Help
//Title: tutor8XX.stl
//Function: Simulation file for the PSD8XX Tutorial
//Designed by: Dan Harris
//Design Date: 6-23-98
//Description: This file is intended to be used in the PSDsilosIII environment as a
// stimulus file for the PSD8XX Tutorial. The idea of this file is not
// to show how the Verilog-HDL language works, but rather the format of
// a .stl file, and how it applies to this tutorial example.
// The main parts of this file are:
// * Parameter declarations which make the file more readable
// * Read, write and "PSEN/" bus cycle tasks for the 80C31
// * An area where the user may wish to add to the file in order to
// test more functions
// * The actual stimulus of the design
//++
// Parameters declarations for the address offsets for the CSIOP address space
//++
For Help, press F1
Project: tutor8xx Part: PSD813F1 02/04/99 12:43:50

4. At this point, you could edit the stimulus file, but since we have a complete stimulus file, click on **LogicSim** to invoke the PSDsilosIII simulator.

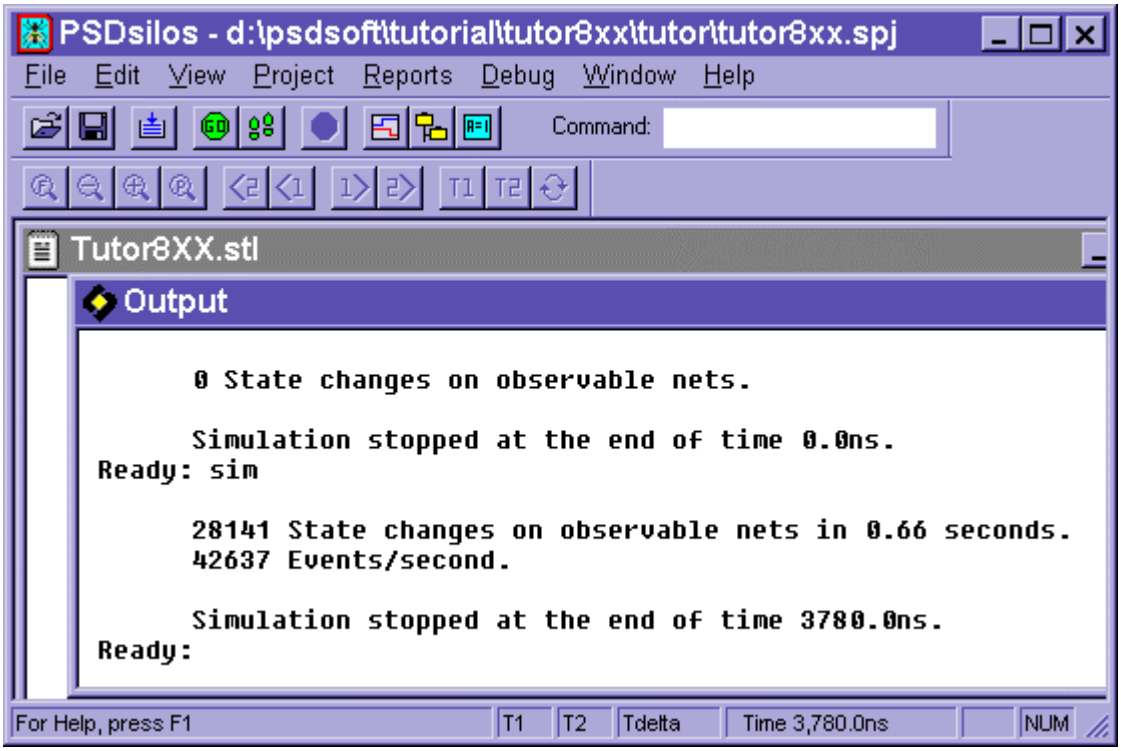
Note: to go directly to the PSDsilos simulator, click on the "Logic Sim" box in the design flow.

The following events happen automatically as a result of clicking on the **LogicSim** button:

- The PSDsilosIII simulator automatically starts
- The simulator automatically loads the project *tutor8xx.spj*, *PSDsoft.run*, and a window displaying the *tutor8xx.stl* file, as shown.


PSDsilos - d:\psdsoft\tutorial\tutor8xx\tutor\tutor8xx.spj - [Tutor8XX.stl]
File Edit View Project Reports Explorer Debug Options Window Help
Command:
1 //Title: tutor8XX.stl
2 //Function: Simulation file for the PSD8XX Tutorial
3 //Designed by: Dan Harris
4 //Design Date: 6-23-98
5 //Description: This file is intended to be used in the PSDsilosIII environment as a
6 // stimulus file for the PSD8XX Tutorial. The idea of this file is not
7 // to show how the Verilog-HDL language works, but rather the format of
8 // a .stl file, and how it applies to this tutorial example.
9 // The main parts of this file are:
10 // * Parameter declarations which make the file more readable
11 // * Read, write and "PSEN/" bus cycle tasks for the 80C31
12 // * An area where the user may wish to add to the file in order to
13 // test more functions
14 // * The actual stimulus of the design
15
16 //+++
17 // Parameters declarations for the address offsets for the CSIOP address space
18 //+++
19
20 //Port A
For Help, press F1
T1
T2
Tdelta
NUM

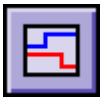
5. Click on the "Go" button, which automatically opens an "Output" window (shown below) for viewing the results of the simulation:

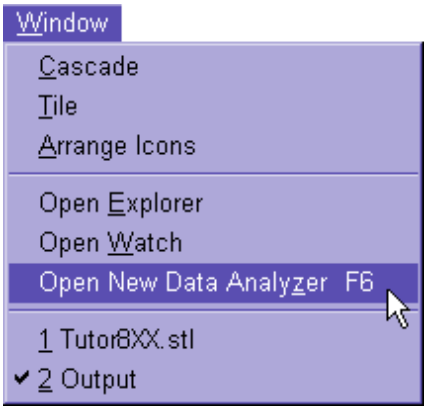


5.5.3 Running the Analyzer

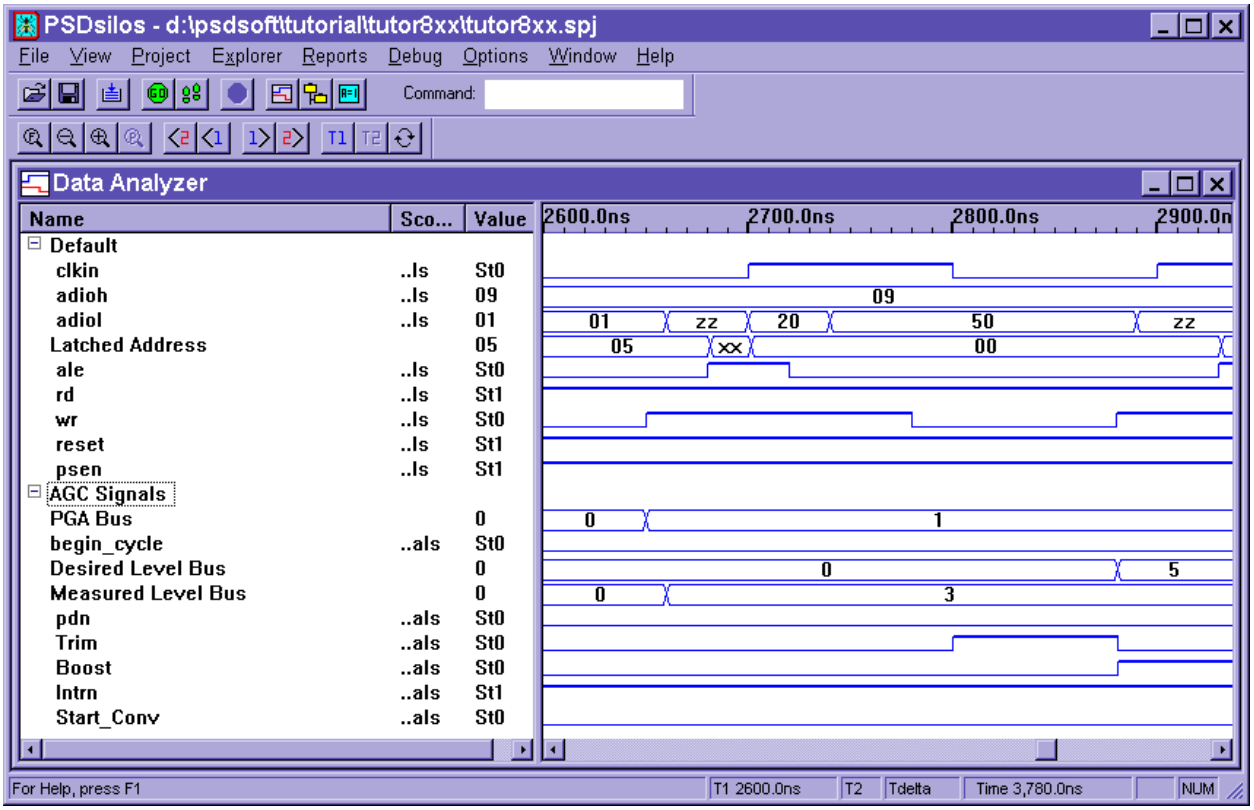
Now that the logic simulation is complete, the results can be displayed with the PSDsilosIII Data Analyzer by performing the following steps:

1. Pull down the **Window** menu and select **Open New Data Analyzer**, press F6, or click on the appropriate button on the tool bar.





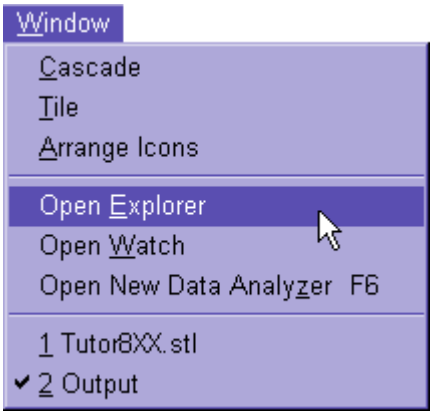
2. The PSD SilosIII Data Analyzer window appears with the simulation results displayed on screen. **Note**: Your screen will look different. See the tutorial on the Data Analyzer and the Explorer under **Help->Contents** on how to rearrange and group signals.



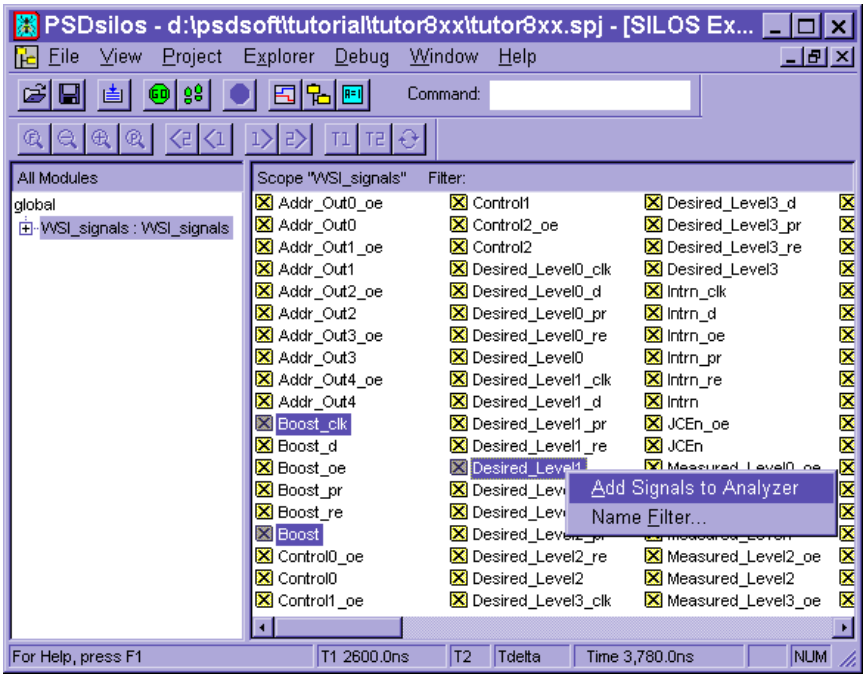
5.5.4 Working With the Explorer

The Explorer in PSD SilosIII can be used in conjunction with the Data Analyzer to add and trace signals.

To open the explorer, ensure that you have simulated the design by following the steps in the "Running the Logic Simulator" section (5.5.2). Next, click on the **Window->Open Explorer** menu selection or the Explorer button and the Explorer window will appear. The Explorer shows all viewable signals.



Signals can be added to the Data Analyzer window using the Explorer by holding the 'CTRL' button down, and clicking on all the signals that you want to add to the Data Analyzer window. Once you have chosen all the desired signals, right-click on one of the signals, and select **Add Signals to Analyzer**. Next, click anywhere in the Data Analyzer window, and the signals you added will appear at the bottom of the window.



For more information on the Explorer or Data Analyzer, see the on-line help and the *PSDsilosIII User Manual*. Also, refer to this manual for information on how to us the PSDsilosIII Watch Window, which is not covered because it is beyond the scope of this tutorial.

5.6 Programming the PSD813F

Programming the PSD813F using PSDsoft can be accomplished two ways: using the parallel programmer called PSDpro or the serial programmer called Flashlink. Section 5.6.1 is dedicated to the PSDpro, and section 5.6.2 is for the Flashlink. Either interface can perform the following operations:

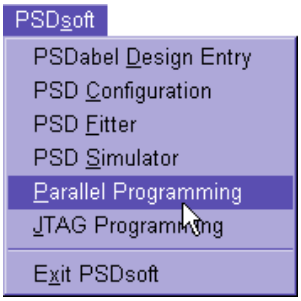
- **Blank Test**—check to see if the device is blank.
- **Upload**—upload the contents of the device.
- **Program**—program the device with the .obj file.
- **Verify—**verify the programmed device against the .obj file in the buffer.
- **Erase—**completely erase the device.

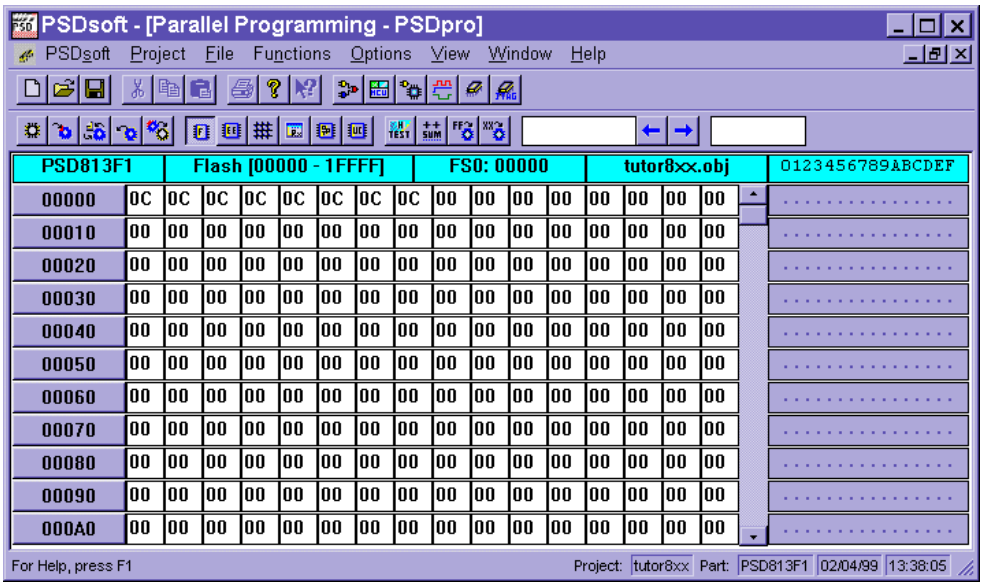
5.6.1 PSDpro

If you have a PSDpro, you should read this subsection. If you only have a Flashlink cable, proceed to the next subsection.

First, let's start the programming interface software within PSDsoft:

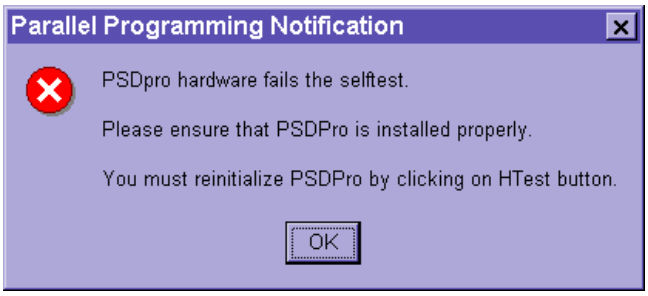
Pull down the **PSDsoft** menu in the main PSDsoft window and choose **PSD Programmer**, click the appropriate button on the tool bar, or click on the "Device Prog" box in the design flow. Any of these actions will open the *tutor8xx.obj* file, as shown below.





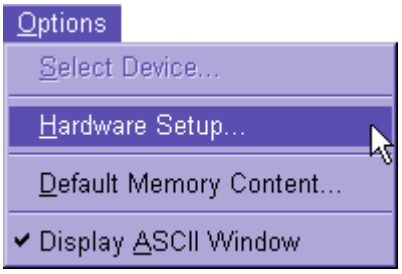
When the object file is opened, you can view the contents of the Flash and EEPROM, the PLD fusemap, and the configuration bits (ACR).

Note: if you don't have your PSDpro properly connected to your parallel port, or if you do not have your port set up correctly (either in Windows or in the BIOS), you may get this error upon entering the program:


Parallel Programming Notification
PSDpro hardware fails the selftest.
Please ensure that PSDPro is installed properly.
You must reinitialize PSDPro by clicking on HTest button.
OK

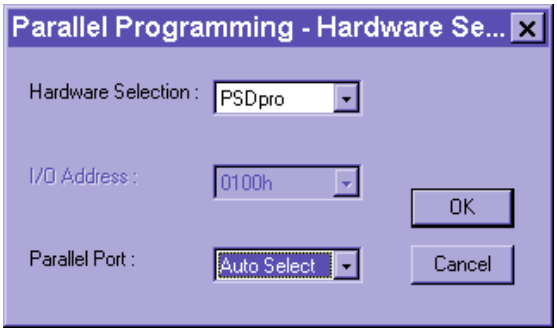
To correct this problem, refer to your computer hardware user manual and the user manual for MS Windows or Windows NT. If you have your PSDpro properly connected to the parallel port, you should see the red power light come on upon entering the Parallel Programming software. If your green "good" light is on, you may skip to the programming portion of this section. If the red power light is on, but the green "good" light is not, take the following steps to set up the PSDpro"

Ensure that your PSDpro is connected to your PC's parallel ports, you can select and configure it by going to the **Options** menu in the PSD Programmer environment and select **Hardware Setup…**.



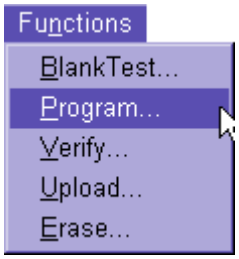
Once the "PSD Programmer – Hardware Setup" dialog box appears, under the "Hardware Section:", select PSDpro.

Next, you will see that the **Auto Select** option becomes active. This means that PSDsoft will automatically detect which PC parallel port your PSDpro is connected to. Just click **OK** and the PSDpro will be detected and configured if the connections are good. You should now see that the green "good" LED is on. If you get a self-test failure like the one shown above, there is most likely a problem with your parallel port. Getting the parallel port working correctly is beyond the scope of this document.



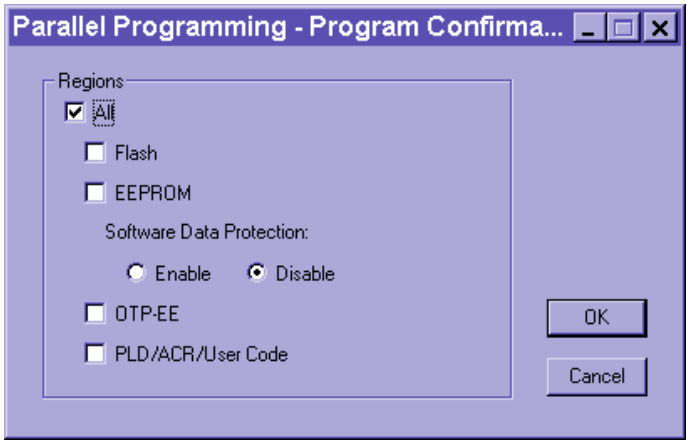
To go any further, you must have your PSDpro working properly, and a PSD813F device ready for programming. Assuming this is the case, take the following steps to program the device using a PSDpro programmer:

1. Pull down the **Functions** menu and select **Program** or click the Program button on the tool bar that's available when the PSD Programmer is invoked.



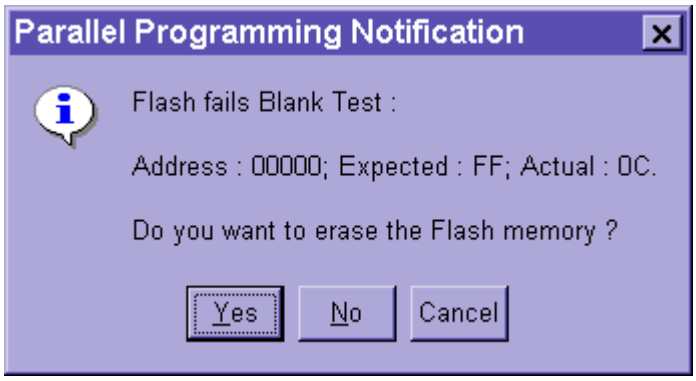
2. The "PSD Programmer – Program Confirmation" dialog box appears, which enables the user to program the Flash, EEPROM or PLD/ACR (PSD Configuration) regions of the device. Select "All", as shown.

3. You may choose to enable Software Data Protect (SDP) for the EEPROM. By doing so, the MCU will have to "unlock" the EEPROM (just like Flash) before writing or erasing. Devices are shipped from the factory with SDP disabled. See the PSD813F Family Data sheet for details.

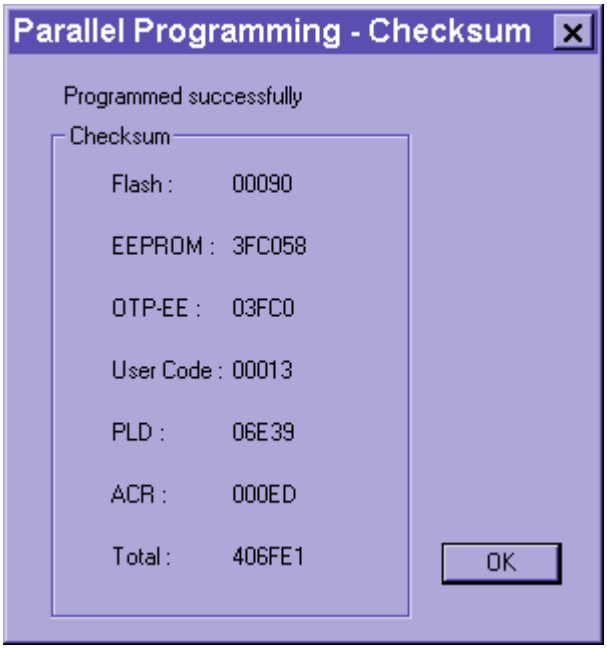


4. Place the PSD device into the programmer using the correct orientation, and snap the lid down on the device carrier. Then, click on the **OK** button. As programming takes place, the PSDpro programmer checks each location after it is programmed to make sure it matches the .obj file contents. If a particular location cannot be programmed properly, an error message will appear. If this occurs, you must start over and program a fully erased and functional part.

Note: if your device is not blank, you will get the following message (or a similar one) after you click **OK**. Put in a blank part or proceed as desired.

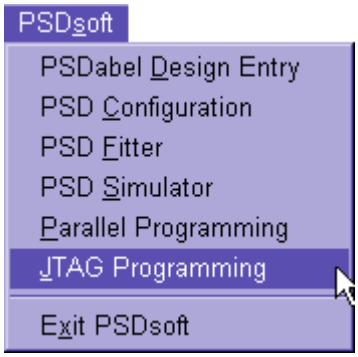

Parallel Programming Notification
Flash fails Blank Test :
Address : 00000; Expected : FF; Actual : 0C.
Do you want to erase the Flash memory ?
Yes
No
Cancel

Once the programming operation has completed, assuming there were no errors, you should see a message similar to the following. (If you used the *tutor8xx.obj* file, you should see the same checksum numbers.) Congratulations, you have now successfully programmed a part. You are now ready to proceed with your own design.



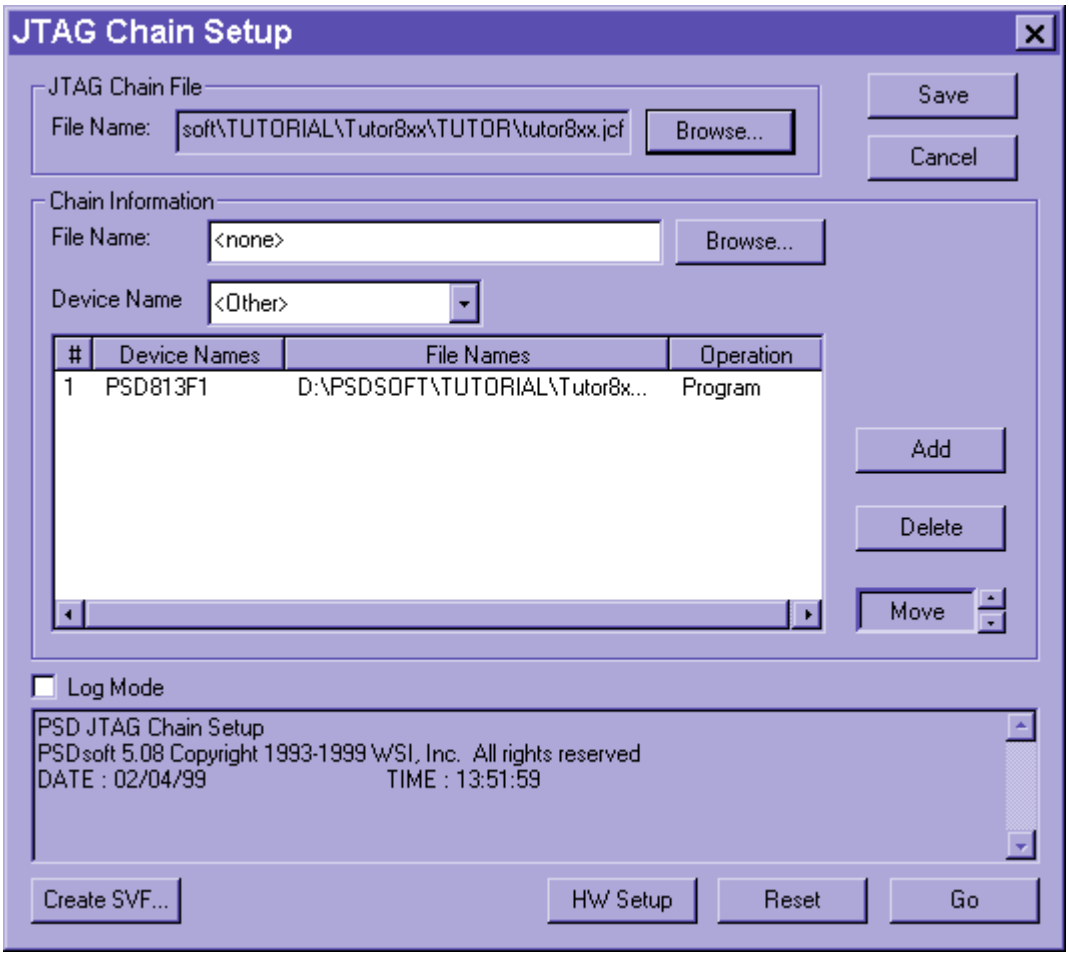
5.6.2 JTAG—FlashLink

If you intend to use JTAG to program your PSD, you should read this section. **Note: before you proceed, ensure that you have the latest version of PSDsoft by visiting our web site—http://waferscale.com.** The first part deals with setting up your FlashLink cable. If you have already confirmed that your FlashLink cable is working properly, you can skip to the second part that deals with the programming process. Before you proceed, you should start the JTAG Programmer under PSDsoft. To do so, pull down the **PSDsoft** menu and choose **JTAG Programming**, click the "JTAG" button on the toolbar, or click on "JTAG Prog" in the design flow.



You should now see the following screen.

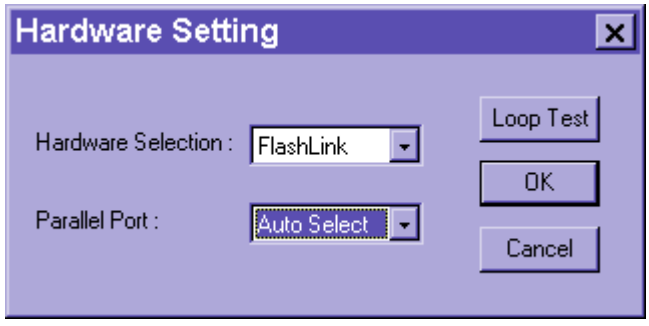
Figure 5—JTAG Chain Setup Dialog Box



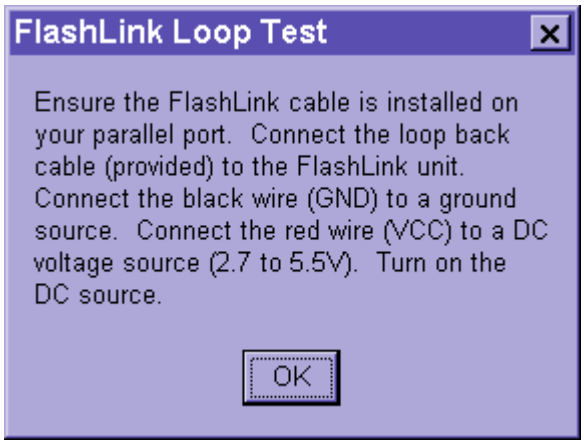
Setting up and Testing your FlashLink Cable.

Ensure that the FlashLink cable is installed on one of your PC's parallel ports (i.e. LPT1). Note: you must also ensure that the parallel port that the FlashLink is connected to is set up correctly in your BIOS and in Windows (95/98 or NT). Consult the user manuals that came with your machine and your operating system to ensure that your parallel port is setup correctly. To setup the FlashLink cable, take the following steps:

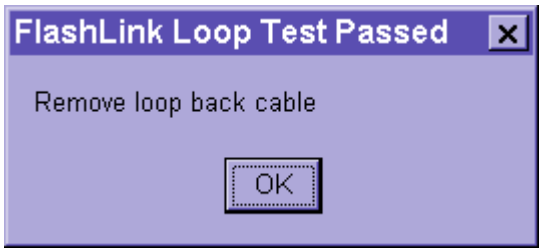
1. Click the **HW Setup** button at the bottom of the "JTAG Chain Setup" dialog box. This brings up the "Hardware Setting" dialog box, as shown. Since FlashLink is currently the only device, this will be shown in the "Hardware Selection:" box. For the "Parallel Port" box, try "Auto Select" first. If Auto Select doesn't work, try selecting the port that you have the Flashlink connected to.



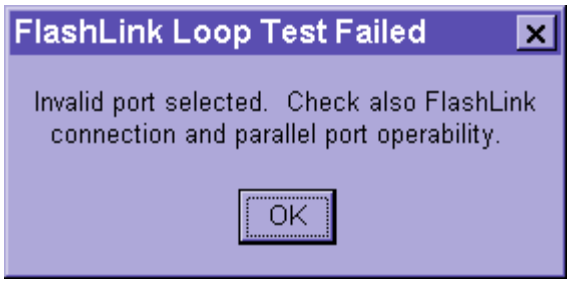
2. We will now perform the Loopback Test. To do so, click on the **Loop Test** button. The "FlashLink Loop Test" information screen will pop up. Follow the instructions for the Test, and click **OK**.


FlashLink Loop Test
Ensure the FlashLink cable is installed on your parallel port. Connect the loop back cable (provided) to the FlashLink unit. Connect the black wire (GND) to a ground source. Connect the red wire (VCC) to a DC voltage source (2.7 to 5.5V). Turn on the DC source.
OK

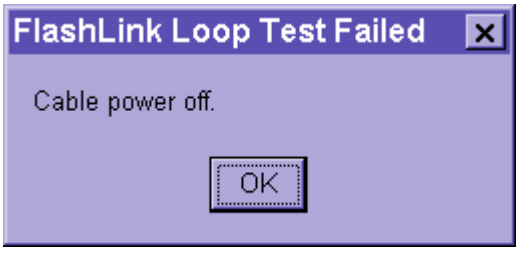
If you get this message, your parallel port is working properly. Click **OK**, and go on to the next section.



If you get this message, the FlashLink is not receiving information from the parallel port. Try changing your selection in the "Hardware Setting" dialog box under "Parallel Port:" to whichever parallel port number you have your FlashLink cable connected, and rerun the test above. If you are still experiencing problems, you will need to get your parallel port working properly before you can continue, which is beyond the scope of this document.



If you get this message, ensure that Vcc and Ground wires are connected to the correct pins and that the pins are at the correct voltage potential. You should have between 4.5 and 5.5 V for non low-power parts (parts with no V suffix) and between 2.7 and 3.3 V for low-power parts with the V suffix. If you have the correct voltage, you should not see this message.



Once you have determined that your Parallel port is working properly (because your FlashLink Loop Test passed), you are now ready to attempt to program your part. The next part covers how to program the part using your FlashLink cable.

Programming the PSD813F via JTAG using a FlashLink Cable

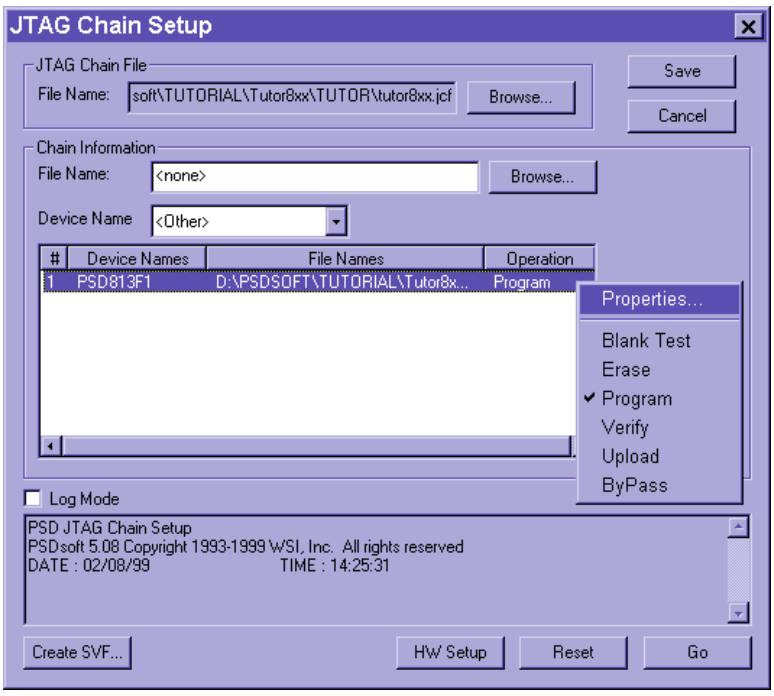
This part of the section covers how to program your PSD813F via JTAG using a FlashLink cable. This part assumes you have tested your FlashLink Cable and it has passed the Loopback Test in the previous part.

Note: if you encounter any problems during programming and you have passed the Loopback Test described above, your problem may be due to one of several conditions, including:

1) You don't have the PSDsoft version 5.07 or later. If this is the case, go to our web site—http://waferscale.com—and download the latest version.
2) Your PSD813F part has been damaged. You can easily check this if you have another part to try to program. If you get the same error, chances are you have a problem with your JTAG connection (covered last).
3) You have previously programmed the PSD813F with a .obj file that disables the JTAG pins. You must either erase the PSD using the PSDpro, or the MCU must enalbe the pins at run-time using the JTAG register. Refer to the PSD813F family data sheet for details.
4) You are having a problem with your JTAG connection. This problem is beyond the scope of this document. Refer to our Application Note 54 for more information on JTAG. If you can't solve your problems with this Application Note 54, you can go to the Technical Support section of our web site and email a WSI Applications Engineer from there.

You should now be back in the "JTAG Chain Setup" dialog box (Figure 5). The JTAG Programming software should have automatically loaded the *tutor8xx.jcf* file. To see how to create your own JTAG chain file, see the subsection "Creating a JTAG Chain File." To program your PSD813F using a FlashLink cable, take the following steps:

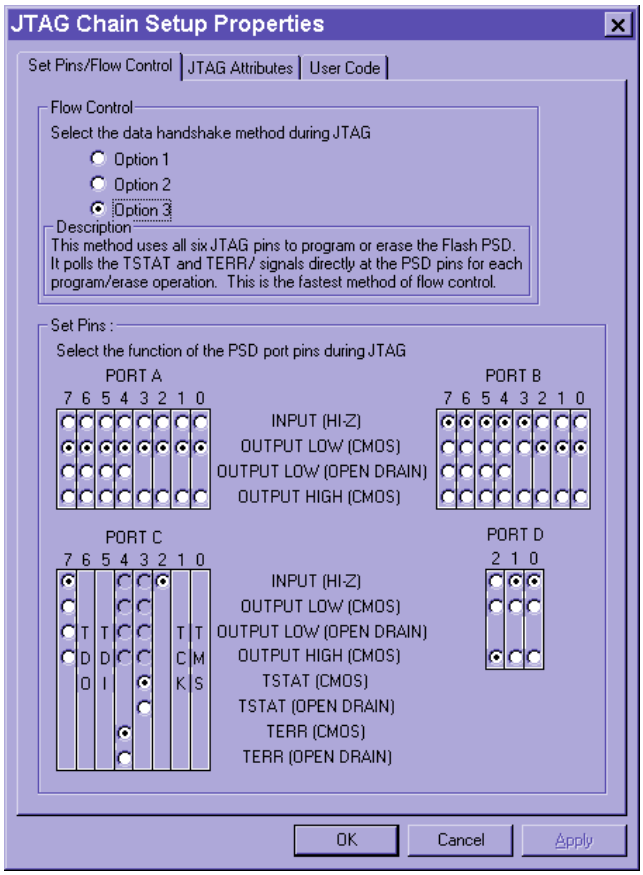
1. Right-click on line # 1 in the "Chain Information" box and select **Properties …**, as shown.



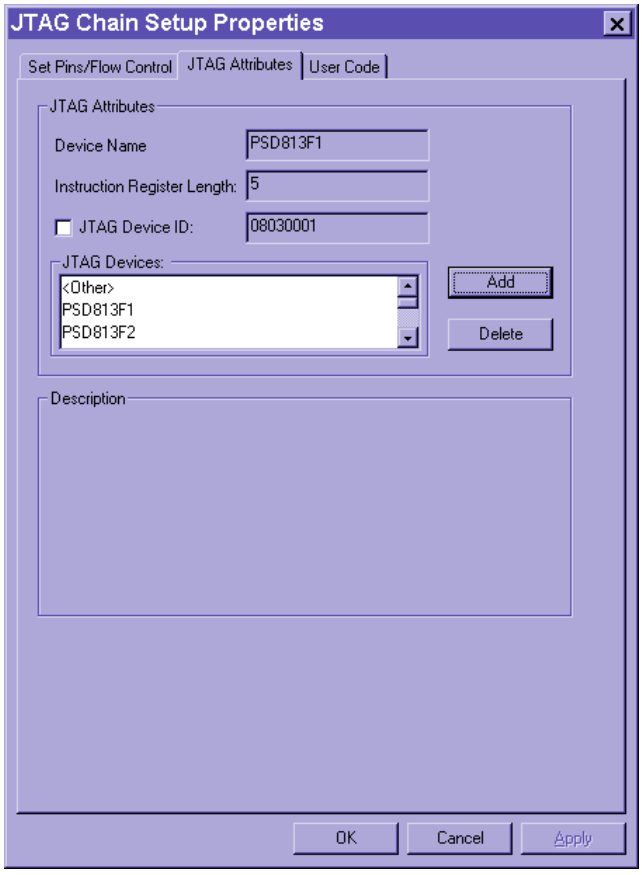
2. You should see the "JTAG Chain Setup Properties." Ensure that the "Set Pins/Flow Control" tab is selected, and set up the window with the following selections based Figure 2 of this tutorial (proper selections shown right):
   - Under "Flow Control", select **Option 3**.

   In the "Set Pins" box, set up the ports as follows:
   - Port A: set all the pins to "OUTPUT LOW (CMOS)"
   - Port B: set pins pb7 to pb3 to "INPUT (HI-Z)", and pins pb2 to pb0 to "OUTPUT LOW CMOS)"
   - Port C: change pc3 to "TSTAT (CMOS)", and pc4 to "TERR (CMOS)". Leave the rest of the pins as is.
   - Port D: set pins pd1 and pd0 to "INPUT (HI-Z)", and pd2 to "OUTPUT HIGH (CMOS)"

   Click on the **Apply** button if you made any changes. This saves any changes you have made so far.

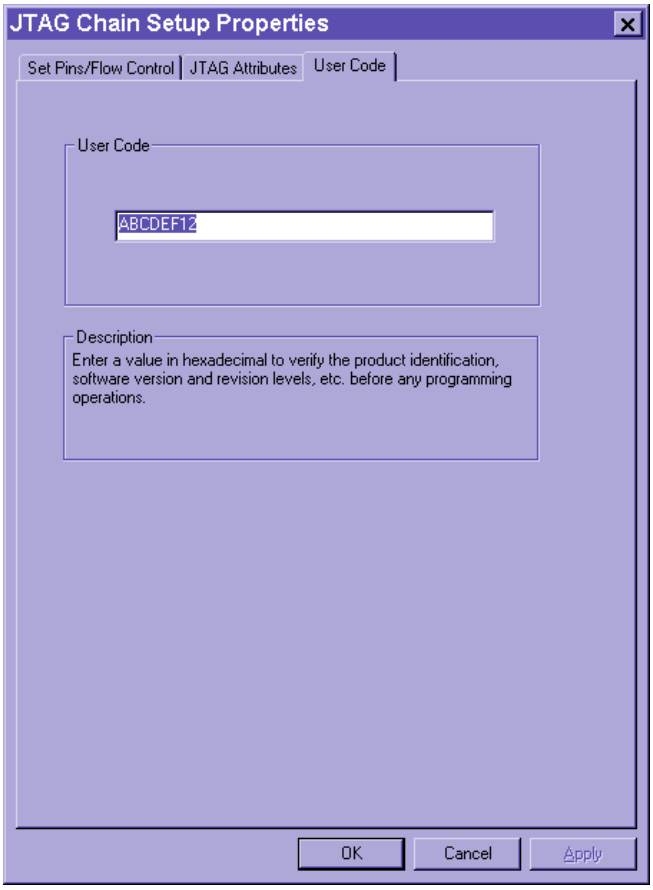


3. Click the "JTAG Attributes" tab. You will see the screen shown. If you are using the *tutor8xx.jcf* file, there is nothing for you to do here. If you are using your own JTAG Chain File, read the following note and go to the appropriate section below.

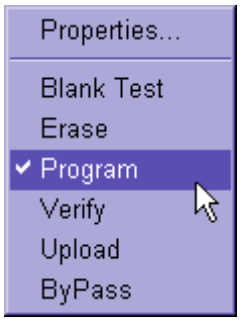


Note: the "Device Name", "Instruction Register Length:", and "JTAG Device ID:" are all grayed out because this information is automatically entered whenever you select a PSD813F device. If you wanted to enter information about a non-PSD device that would be included in your chain, here is the place to do it. If you did add another device, you would need to enter valid information in the "JTAG Attributes" section. Also, note that if you select the "JTAG Device ID:" box, PSDsoft will verify the JTAG ID before programming or erasing the device.

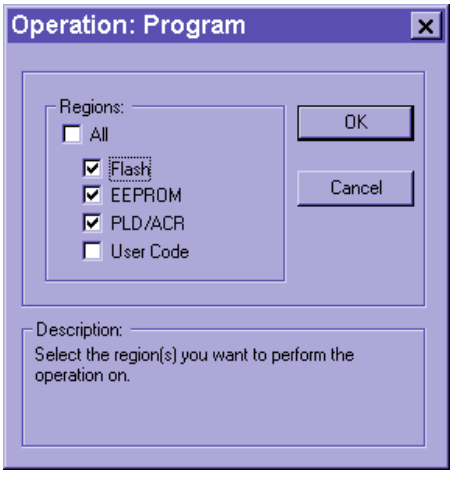
4. Click on the "User Code" tab. If you enter a value in the "User Code" box, the value will be compared with the User Code already programmed into the device before any JTAG operation occurs (e.g. Erase, Program, etc.). If you leave this area blank, no comparison will be done. Enter `ABCDEF12` in the "User Code" box. Then press **Apply** (which grays the "Apply" button out), and finally press **OK**.



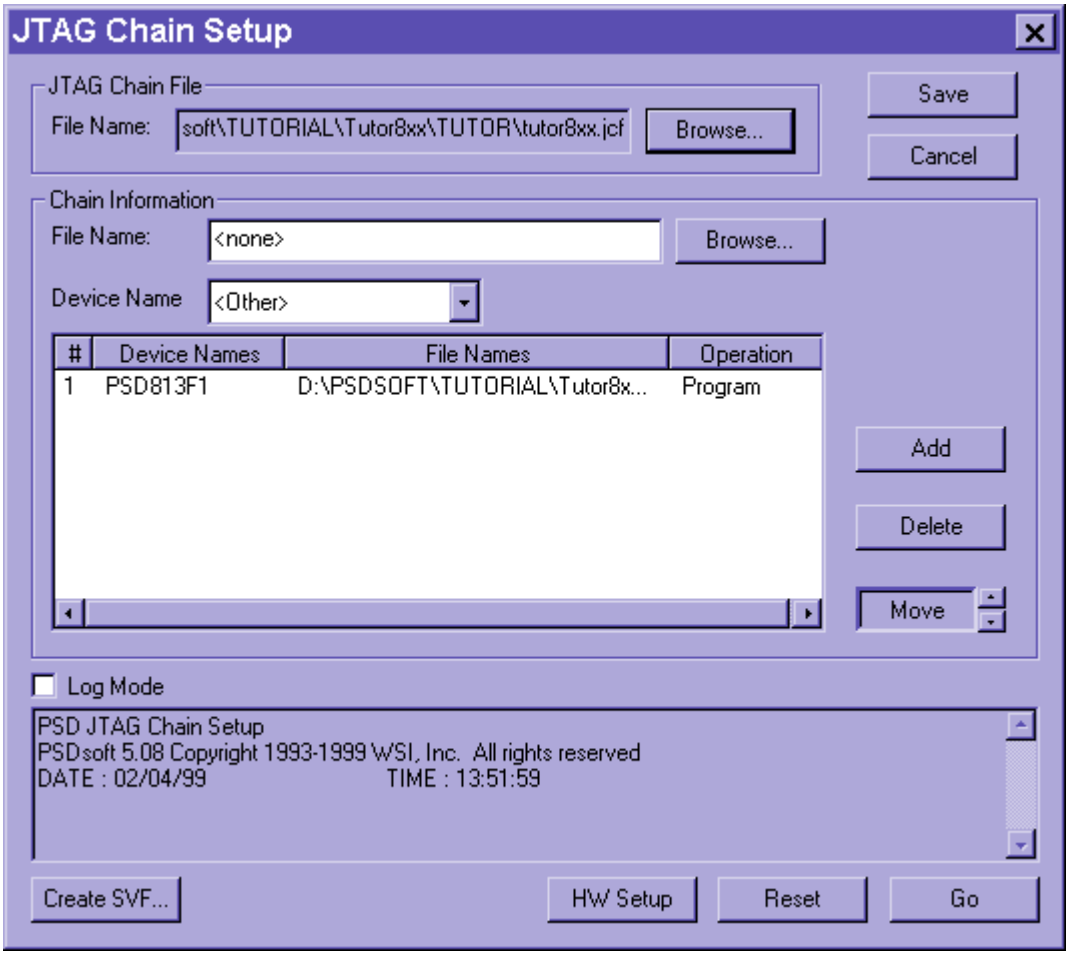
5. Now, you should be back to the "JTAG Chain Setup" window (Figure 5). Right-click on the same line as you did in step 1, only this time, choose **Program**.



6. You should now see the "Operation: Program" dialog box. Ensure that **All** is checked in the "Regions:" section.



7. Your "JTAG Chain Setup" should now have the following look. Select **Go**, and the PSD will be programmed with the information in the *tutor8xx.obj* file.



Note: during the program operation, PSDsoft holds the reset (Rst) line on the FlashLink cable low. Your MCU should be connected to this Reset line so that it does not try to execute code during a JTAG programming operation. The Rst signal output from the FlashLink cable is open-collector and pulled up in the cable. Once finished with the Program operation, PSDsoft releases the Reset line on the FlashLink cable.

Congratulations, you have now programmed a PSD813F part via JTAG using a FlashLink cable! You are now ready to proceed with your own design. Be sure and read the next subsection if you are unsure as to how to set up a JTAG chain file.

5.6.2.1 Setting up a JTAG Chain

The following rules apply for setting up a JTAG chain:

- A JTAG chain of one to or more devices must be defined.
- All JTAG compatible devices that are connected to the JTAG bus, including the PSD813F and non-PSD devices from other vendors compose a JTAG chain.
- Non-PSD devices that are part of the JTAG chain will be placed in bypass mode automatically.
- The length of the instruction register, along with a name and device ID must be entered for each non-PSD device. (In future versions of PSDsoft, you will be able to automatically load this information with a BSDL file.)
- Before programming the PSD device(s), the user must have a valid .obj file for each PSD device in the chain.

Please refer to document titled *JTAG Information - PSD8xxF* (*Application Note 54*) for information in these areas:

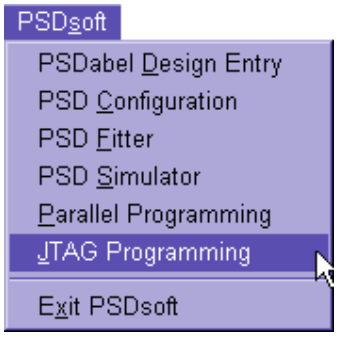
- JTAG Spec Compliance
- Programming Support
- Program/Erase Flow Control
- SVF/BSDL file information
- Enhanced ISP functions
- Multiplexed JTAG pin functions
- Dedicated JTAG pin functions
- WSI JTAG ISP connector
- JTAG Chaining
- Electrical Considerations.

Now, let's step through a sample JTAG chain setup, and create a JTAG chain file (.jcf). The following are the steps you would take:

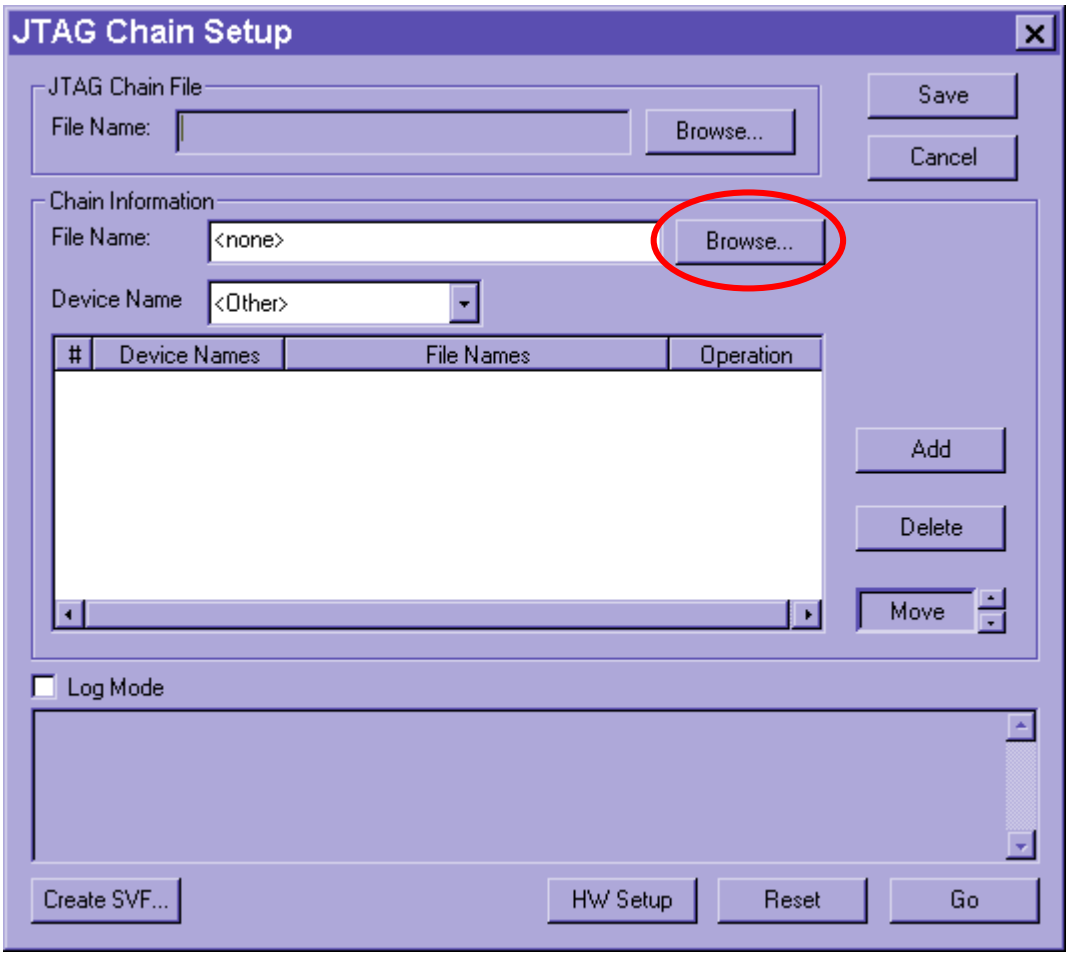
1. If you are currently using the JTAG Programming software, exit it by clicking on the button in the upper right-hand corner of the "JTAG Chain Setup" dialog box.

2. Erase the *tutor8xx.jcf* file from the \PSDsoft\Tutorial\Tutor8xx\Tutor directory.

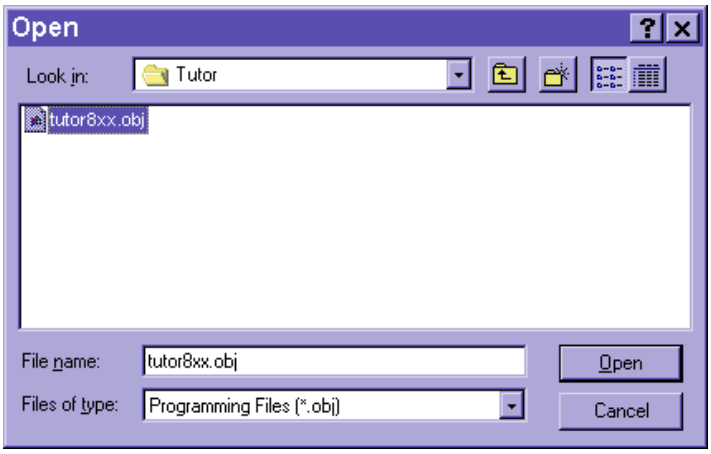
3. Start the JTAG Programming program in PSDsoft. (See the beginning of Section 5.7.2.)



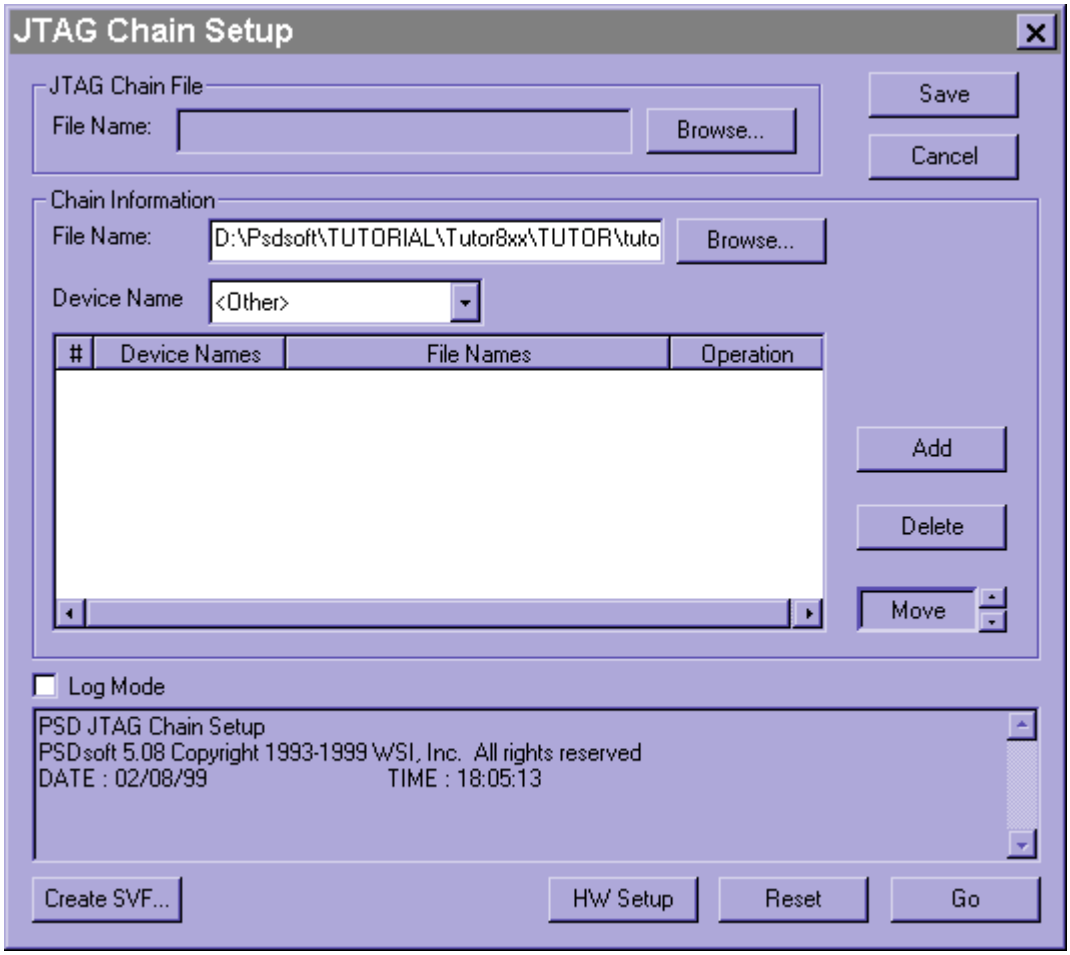
4. You should see a "JTAG Chain Setup" dialog box that is "blank," as shown below. Go to the "Chain Information" box and click the **Browse…** button.



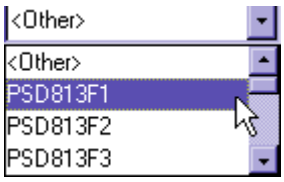
5. The "Open" window pops up. Select the *tutor8xx.obj* file and click **Open**.



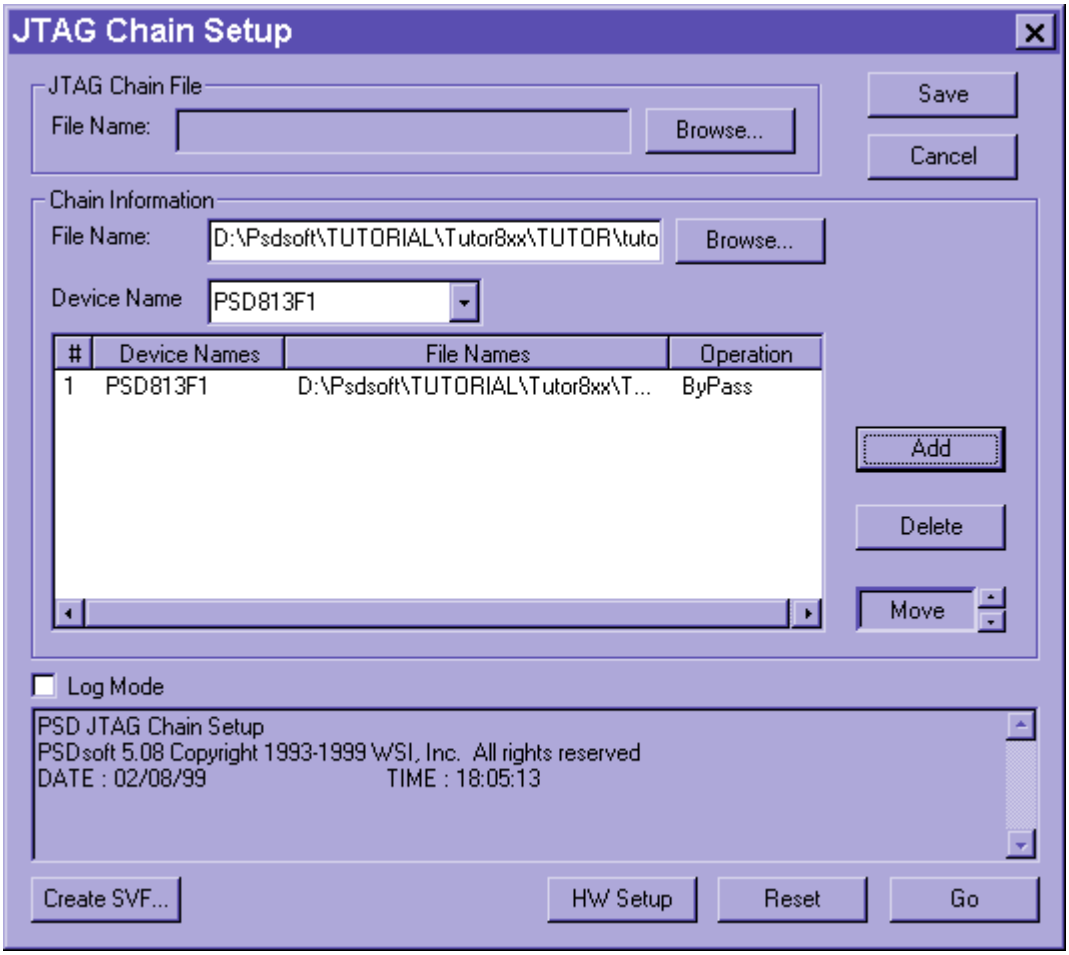
6. Your “JTAG Chain Setup” window should now look like this:



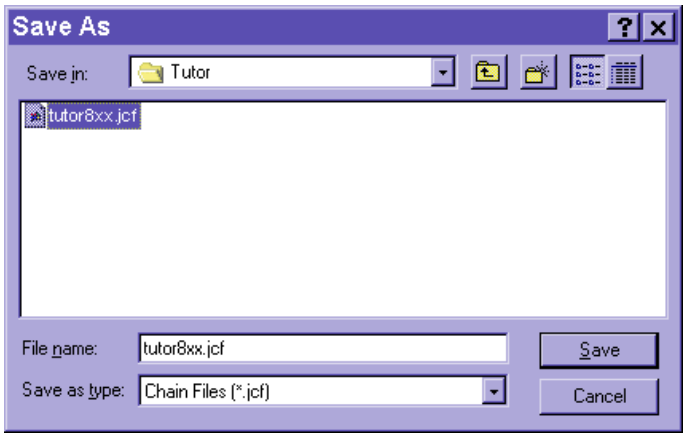
Note: If the device name (PSD813F1 in this case) does not automatically appear in the “Device Name” window, click on the down arrow next to “Device Name”, and select the PSD813F1.



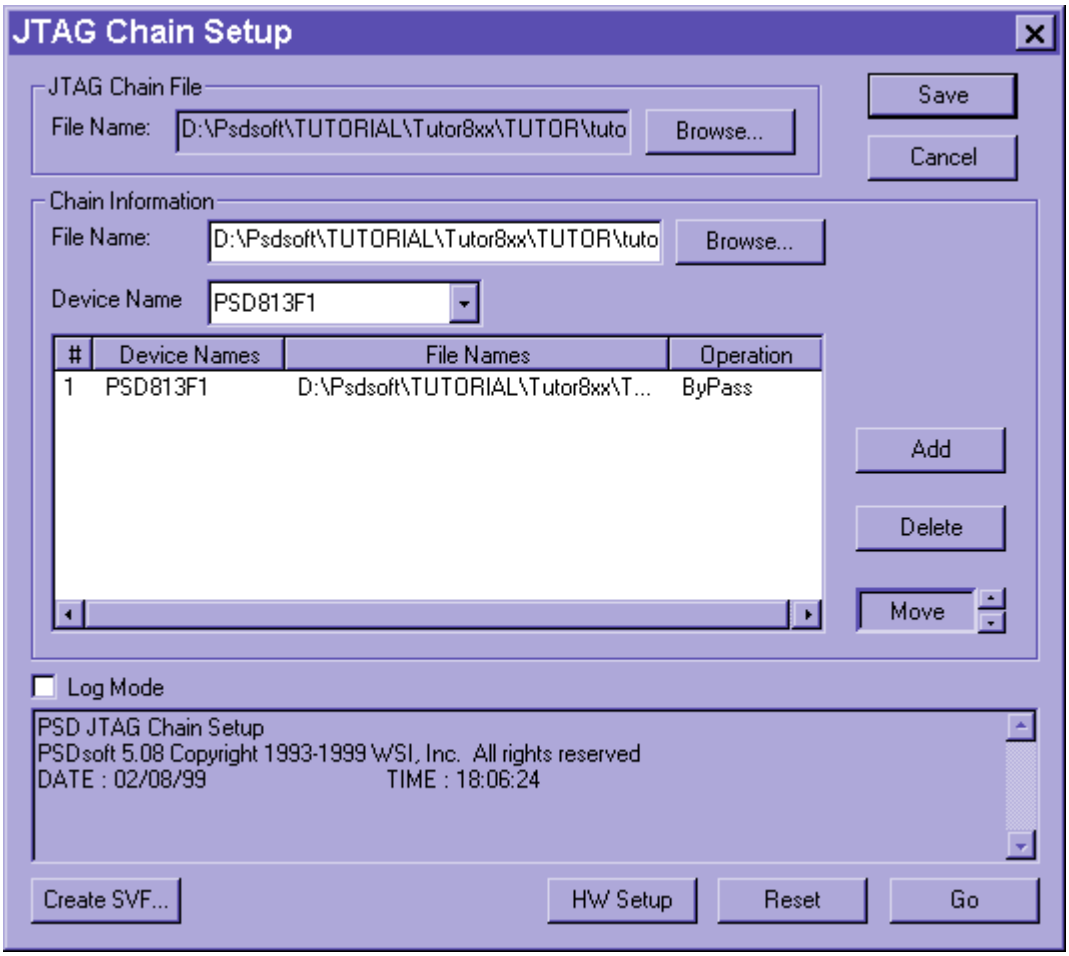
7. Click the **Add** button. Your "JTAG Chain Setup" window should look like this:



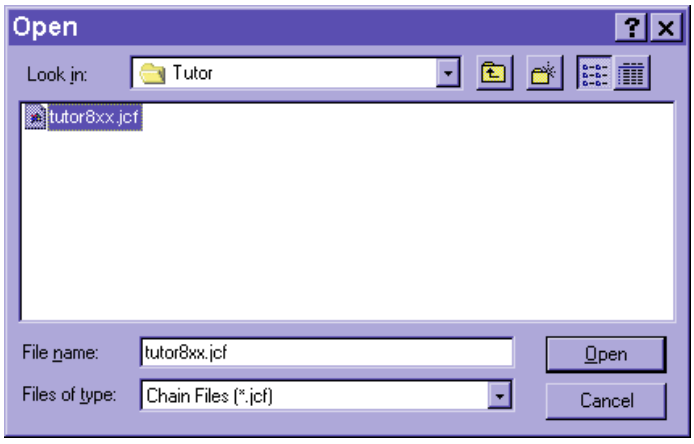
8. Save your work in a JTAG Chain File for future use. To do so, click on the **Save** button. This action brings up the "Save As" dialog box. Type `tutor8xx` in the "File name:" box, and click **Save**. The file *tutor8xx.jcf* will be created. This is now identical to the file we were working with before.



9. Now, the "JTAG Chain Setup" window should look like this:



Note: if you need to load this .jcf file in the future, you will have to click on the top **Browse…** button, which would bring up the "Open" dialog box. Choose the *tutor8xx.jcf* file, and click **Open**.



10. If you wanted to add any more devices to the JTAG chain file, you would repeat Steps 4 through 8.

6 ISP and the PSD813F

The PSD813F may be programmed in-system, with or without participation from the MCU. For ISP with the MCU, see Appendix F for UART download information and considerations. For ISP without MCU participation, see section 5.6.2 and *Application Note 54, JTAG Information - PSD8XXF* for FlashLink JTAG programming within the PSDsoft environment.

Appendix A—ABEL Design File—*Tutor8xx.abl*

```
module Tutor8xx
title '8xx Tutorial Design File';
// Designed by:  Dan Harris and Mark Rootz
// Design date:  6-16-98
// Description:  This shows the logic implementation of the sample design in the 8xx Tutorial.
//               The design highlights the following functionality of the PSD8xx:
//               * Effective and efficient use of the Input and Output Micro<->Cells
//               * How to use I/O pins while the underlying Micro<->Cell is being used for
//                 other functionality.
//               * Use of the WSIPSD PROPERTY statement to output demultiplexed address bits,
//                 and define Input Micro<->Cells/Output Micro<->Cells.
//               * Multiplexing the JTAG pins with other I/O.
//               * How to logically interface to an 80C31 MCU, RTC, and an AGC circuit.
// Revision:     1.0
// Rev Date:     9-21-98
// Convention:   The n is used throughout the file to indicate active low signals.
//               Note that it is not used with the reserved signal names below.


"*************************    Bus Interface signal declarations    *************************
// The reserved signal names are automatically assigned to the appropriate pin
// The following are inputs from the MCU
wr        pin;    "CNTL0 Input:(pin 47)- write strobe
rd        pin;    "CNTL1 Input:(pin 50)- read strobe
psen      pin;    "CNTL2 Input:(pin 49)- program store enable
ale       pin;    "PD0 Input:(pin 10)- address latch enable
reset     pin;    "Input:(pin 48)- system reset
a15..a0   pin;    "Input:(pins 46..39,37..30)- demuxed address


"*************************     Port A, B, C, D pin declaration     *************************

// Port A I/O
// Control outputs are MCU I/O mode outputs
Control0..Control2        pin 23, 22, 21;          "Some generic control signals
// Assign the latched/demultiplexed address to Port A, pins pa4 to pa0.
WSIPSD PROPERTY 'Address_Out Aout[4:0]:Addr_Out[4:0]';


// Port B I/O
PGA_Din2..PGA_Din0        pin 5, 6, 7;     "Data bits used to program the PGA
                                           "Implemented with MCU I/O mode
Measured_Level3..Measured_Level0 pin istype 'reg';  "Upper 4 bits of the A/D converter (ADC)
WSIPSD PROPERTY 'DataBus_IMC D[7:4]:Measured_Level[3:0] PortB';

// Port C I/O
// Note that pins pc0, pc1, and pc5-6 are multiplexed output/JTAG signals.  pc3, pc4, and pc6
// are JTAG signals that are not multiplexed.
// Ensure that under "Global Configuration" with the "JTAG Configuration" tab selected that none
// of the boxes enabling various JTAG signals on certain pins are checked because the device
// will expect only valid JTAG signals on these pins, and no multiplexing can be done under these
// circumstances.
// Pin pc2 (pin 18) is used for VSTBY (set in global configuration)
Intrn             pin 20;  "Interrupt the MCU when the gain needs to be changed/JTAG TMS
Start_Conv        pin 19;  "Start Conversion signal for the ADC/JTAG TCK
Trim              pin 17;  "The gain is too high and needs to be decremented/JTAG TSTAT
Boost             pin 14;  "There is not enough gain--increment it/JTAG TERRn
JCEn              pin 11;  "JTAG chip enable signal used to demultiplex Port C output and JTAG I/O


// Port D I/O
// pd0 (pin 10) is assigned above to the ALE signal from the microcontroller.
// Any external chip selects that are generated by decoding an address should be placed on
// Port D when possible to save as many resources as possible.
RTCcsn   pin 8;   "Real Time Clock (RTC) chip select/JTAG TDO
clkin    pin;     "Port D pin pd1 (pin 9) System clock


// Output Micro<->Cell assignments
WSIPSD PROPERTY 'DataBus_OMC D[7:4]:Desired_Level[3:0] MCELLAB';
WSIPSD PROPERTY 'DataBus_OMC D7:begin_cycle MCELLBC';

"*************************        Internal node declarations       *************************

mxord3                    node;    "This signal is needed to save product terms
meqd                      node;    "True when the measured signal equals the desired signal level
```

```
begin_cycle              node istype 'reg';       "This signal takes the state machine out of idle
STATE1..STATE0           node istype 'reg';       "State machine bits
Desired_Level3..Desired_Level0    node istype 'reg';       "The desired gain level

fs7..fs0         node;    "Main Flash memory segments
ees3..ees0       node;   "EEPROM memory segments
// Reserved node names
rs0                      node;    "Select for the SRAM memory space
csiop                    node;   "Control register
jtagsel          node 102;        "This is the JTAG enable product term.  It is used to enable
                                  "the JTAG port signals.
pgr1..pgr0               node;    "Internal PSD Page Register bits


// The following page register bit definitions are an example of how to manipulate memory to
// facilitate ISP. This scheme is explained in Appendix F of Application note 57.


swap     node 117;        " This page register bit (pgr7) will be used for swapping
                          " memory segments after a firmware download from the 8031
                          " UART port has completed.  When swap = 0, the secondary
                          " NVM occupies boot area for ISP, swap = 1, primary NVM
                          " occupies boot area.

enable_data_half node 116;
                          " This page register bit (pgr6) will be used to manipulate
                          " the EEPROM.  The use of this bit is one way to divide the
                          " EEPROM in into two equal sections, one for boot and one for
                          " general data. When this bit=0, the boot section is active.
                          " When this bit = 1, the data section is active.


"*************************              DEFINITIONS              *************************

DLEVEL = [Desired_Level3..Desired_Level0]; "Desired gain level set by MCU
MLEVEL = [Measured_Level3..Measured_Level0];       "Measured gain level latched by IMCs
STATE_MACHINE = [STATE1..STATE0];
X = .x.; "Don't care symbol
C = .c.; "Clock symbol
page = [pgr1,pgr0];
address = [a15..a0];      "De-muxed microcontroller address signals


EQUATIONS

"*************************            DPLD equations            *************************

// Generate active high chip selects for the main Flash segments.  Each segment is 16K bytes
// for the PSD813FX devices.
fs0 = ((address >= ^h8000) & (address <= ^hBFFF) & (page == 3) & !swap)
    # ((address >= ^h0000) & (address <= ^h3FFF) & (page == X) &  swap);
fs1 =  (address >= ^h4000) & (address <= ^h7FFF) & (page == X);
fs2 =  (address >= ^h8000) & (address <= ^hBFFF) & (page == 0);
fs3 =  (address >= ^hC000) & (address <= ^hFFFF) & (page == 0);
fs4 =  (address >= ^h8000) & (address <= ^hBFFF) & (page == 1);
fs5 =  (address >= ^hC000) & (address <= ^hFFFF) & (page == 1);
fs6 =  (address >= ^h8000) & (address <= ^hBFFF) & (page == 2);
fs7 =  (address >= ^hC000) & (address <= ^hFFFF) & (page == 2);


// Generate active high chip selects for the EEPROM segments.  Each segment is 8K bytes for
// the PSD813F1 devices.
ees0 = ((address >= ^h0000) & (address <= ^h1FFF) & (page == X) & !swap)
     # ((address >= ^h8000) & (address <= ^h9FFF) & (page == X) &  swap & !enable_data_half);
ees1 = ((address >= ^h2000) & (address <= ^h3FFF) & (page == X) & !swap)
     # ((address >= ^hA000) & (address <= ^hBFFF) & (page == X) &  swap & !enable_data_half);
ees2 = (address >= ^hC000)  & (address <= ^hDFFF) & (page == X) &  swap &  enable_data_half;
ees3 = (address >= ^hE000)  & (address <= ^hFFFF) & (page == X) &  swap &  enable_data_half;


//Generate active high chip select for the PSD SRAM (2K bytes).
rs0 = (address >= ^h0100) & (address <= ^h08FF) & (page == X);


// Generate active high chip select for the PSD control registers. 256 contiguous bytes must be
// decoded for all PSD8xx devices.
csiop    = (address >= ^h0900) & (address <= ^h09FF) & (page == X);


// Enable the JTAG port when the JTAG Chip Enable (JCEn) Signal is active
jtagsel = !JCEn;
```

```
"*************************          CPLD equations          *************************

// IMPORTANT NOTE: Comment these next four equations out for the ABEL simulation only.  The
//                 PSDsilosIII Simulator requires the equations (and they are functionally
//                 correct).  The problem is that the MCU presets (loads) and clears these
//                 registers, and the value is not registered through the D input.  However,
//                 the ABEL simulator does not reconize any "dot" extentions (As these would
//                 normally be set up through equations).  The basic functionality can still
//                 be properly tested, but how it is actually implemented in hardware is
//                 slightly different.
//                 So, if you intend to use the ABEL Simulator, comment out the following
//                 four lines so that the test vectors at the end of the file will work
//                 properly.
DLEVEL.ck = 0;
DLEVEL := 0;
begin_cycle.ck = 0;
begin_cycle := 0;

mxord3 = Measured_Level3 $ !Desired_Level3;

// Trim the gain when the Measured signal level is greater than the desired signal level.
// Trim = MLEVEL > DLEVEL
Trim = (Measured_Level3 & !Desired_Level3)
       # ((Measured_Level2 & !Desired_Level2) & mxord3)
       # ((Measured_Level1 & !Desired_Level1) & mxord3 & (Measured_Level2 $ !Desired_Level2))
       # ((Measured_Level0 & !Desired_Level0) & mxord3 & (Measured_Level2 $ !Desired_Level2)
         & (Measured_Level1 $ !Desired_Level1));

// Boost the gain when the Measured signal level is less than the desired one.
meqd = (MLEVEL == DLEVEL);
Boost = !meqd & !Trim;

// Generate the chip select
!RTCcsn = ((address >= ^h0a00) & (address <= ^h0aff));

// Loading of the various registers
MLEVEL.ld = !clkin;

// State machine which controls the conversion start of the ADC, the interrupt to the MCU,
// and the strobing of the IMCs

STATE_MACHINE.ck = clkin;
STATE_MACHINE.re = !reset;

state_diagram STATE_MACHINE;

state 0:
        Start_Conv = 0;
        Intrn = 1;
        if (begin_cycle == 1) then 1 else 0;
state 1:
        Start_Conv = 1;
        goto 2;
state 2:
        Start_Conv = 0;
        goto 3;
state 3:
        !Intrn = Trim # Boost;  "Interrupt when Measured not equal to Desired
        goto 0;


Test_Vectors
// Test the state machine, trim, and boost signals
([clkin, reset, begin_cycle, MLEVEL, DLEVEL] ->
 [Start_Conv, Intrn, STATE1, STATE0, Trim, Boost])
 [ X, 0, X, ^h3, ^h4 ] -> [ X, X, X, X, 0, 1 ];    "system in reset
 [ 0, 1, X, ^h4, ^h4 ] -> [ 0, 1, 0, 0, 0, 0 ];    "system not in reset
 [ C, 1, 1, ^h5, ^h4 ] -> [ 1, 1, 0, 1, 1, 0 ];
 [ C, 1, 1, ^h5, ^h4 ] -> [ 0, 1, 1, 0, 1, 0 ];
 [ C, 1, 1, ^h5, ^h4 ] -> [ 0, 0, 1, 1, 1, 0 ];
 [ C, 1, 1, ^h4, ^h4 ] -> [ 0, 1, 0, 0, 0, 0 ];
 [ C, 1, 0, ^h4, ^h4 ] -> [ 0, 1, 0, 0, 0, 0 ];

end
```

Appendix B—Stimulus File—*Tutor8xx.stl*

The *tutor8xx.stl* file consists of four sections:

1. Parameter Definitions: each of the PSD813F control registers has an I/O address (offset from the CSIOP base address). The parameters make the stimulus file easier to read.

2. User-defined tasks: used to define and implement a microcontroller bus cycles. In each task, the timing of the control signals and address/data bus should follow that of the microcontroller, but don't have to be exact, just to scale. The PSD Simulator will simulate a bus cycle every time read, write, or psen task is called.

3. Signal Initialization: you must specify the initial logic level of all the input signals before simulation. **Note:** The output signals that you want to simulate should be initialized to a high impedance state.

4. The stimulus inputs: here the stimulus inputs are needed to perform MCU read/write bus cycles to access the Flash, EEPROM, SRAM or I/O ports. Inputs can also be generated to exercise the CPLD functions.

```
//Title:         tutor8xx.stl
//Function:      Simulation file for the PSD8xx Tutorial
//Designed by:   Dan Harris
//Design Date:   6-23-98
//Description:   This file is intended to be used in the PSDsilosIII environment as a
//               stimulus file for the PSD8xx Tutorial.  The idea of this file is not
//               to show how the Verilog-HDL language works, but rather the format of
//               a .stl file, and how it applies to this tutorial example.
//               The main parts of this file are:
//               * Parameter declarations which make the file more readable
//               * Read, write and "PSEN/" bus cycle tasks for the 80C31
//               * An area where the user may wish to add to the file in order to
//                 test more functions
//               * The actual stimulus of the design

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Parameters declarations for the address offsets for the CSIOP address space
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//Port A
parameter Port_A_Dir_Reg='h0906,  Port_A_Cntl_Reg ='h0902;
parameter Port_A_Dout_Reg='h0904, Port_A_Din_Reg ='h0900;
parameter Port_A_IMC='h090A,      Port_A_Drive_Sel = 'h0908;
parameter Port_A_En_Out='h090C;

//Port B
parameter Port_B_Dir_Reg='h0907,  Port_B_Cntl_Reg ='h0903;
parameter Port_B_Dout_Reg='h0905, Port_B_Din_Reg ='h0901;
parameter Port_B_IMC='h090B,      Port_B_Drive_Sel = 'h0909;
parameter Port_B_En_Out='h090D;

//Port C
parameter Port_C_Dir_Reg='h0914,  Port_C_En_Out ='h091A;
parameter Port_C_Dout_Reg='h0912, Port_C_Din_Reg ='h0910;
parameter Port_C_IMC='h0918,      Port_C_Drive_Sel = 'h0916;

//Port D
parameter Port_D_Dir_Reg='h0915,  Port_D_Drive_Sel ='h0917;
parameter Port_D_Dout_Reg='h0913, Port_D_Din_Reg ='h0911;
parameter Port_D_En_Out='h091B;

//Port AB OMCs
```

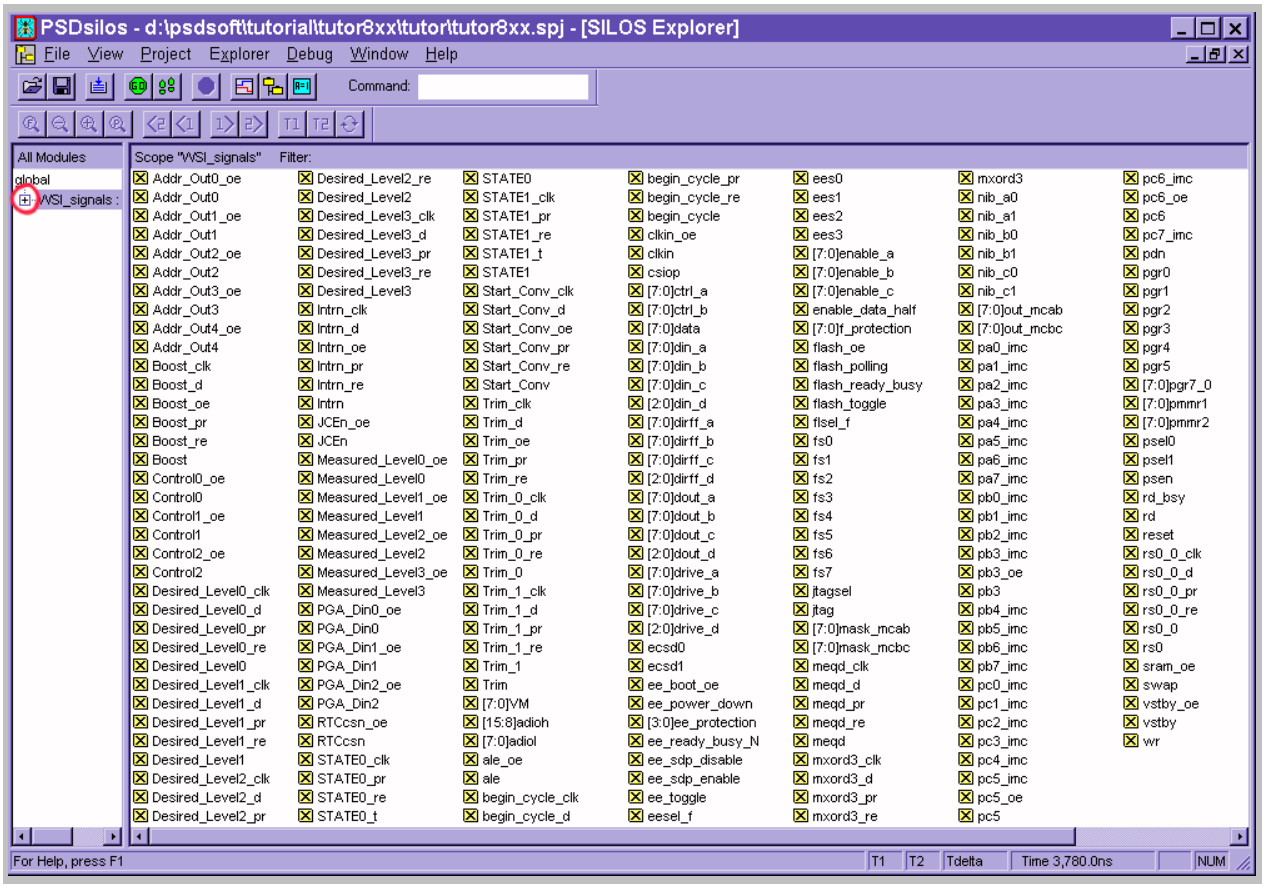
```
parameter Port_AB_OMC='h0920,     Port_AB_OMC_Mask = 'h0922;

//Port BC OMCs
parameter Port_BC_OMC='h0921,     Port_BC_OMC_Mask = 'h0923;

//Other control registers
parameter FLASH_Protect='h09C0,   EEPROM_Protect = 'h09C2;
parameter PMMR0_Reg='h09B0,       PMMR1_Reg ='h09B2;
parameter PMMR2_Reg='h09B4,       JTAG_En = 'h09C4;
parameter Page_Reg='h09E0,        VM_Reg   ='h09E2;


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Defining tasks to simulate 80C31 bus cycles (read, write and psen bus cycles).
// Note that the cycles are shortened for simulation purposes, but the functionality
// remains the same.
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//The "write task" implements the 80C31 write bus cycle

task write;
input [15:0] addr_bus;
input [7:0] data_in;

begin
#20  ale = 1;             //Latch the address lines
#20  adio = addr_bus;     //Read the valid address (adio defined in .top file)
#20  ale = 0;             //Ale inactive
#20  adio[7:0] = data_in; //Write operation
#40  wr = 0;              //Write pulse
#100 wr = 1;              //Write ends
#10  adio[7:0] = Z8;      //Z16 defined in .top file
end

endtask


//The "read task" implements  the 80C31 read bus cycle timing

task read;
input [15:0] addr_bus;

begin
#20  ale = 1;             //Latch the address lines
#20  adio = addr_bus;     //Read the valid address
#20  ale = 0;             //Ale inactive
#20  adio[7:0] = Z8;      //Float address bus (Z8 defined in .top)
#40  rd = 0;              //Read pulse
#100 rd = 1;              //Read ends
end

endtask


//The "psen task" implements  the 80C31 psen program fetch bus cycle

task psen;
input [15:0] addr_bus;

begin
#20  ale = 1;             //Latch the address lines
#20  adio = addr_bus;     //Set-up the right address
#20  ale = 0;             //Ale inactive
#20  adio[7:0] = Z8;      //Float address bus
#40  psen = 0;            //Read pulse
#100 psen = 1;            //Read ends
end

endtask


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Define some busses here to make the program easier to read.
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//adrout is the latched address output on Port A

reg [4:0] adrout;
reg Addr_Out4, Addr_Out3,Addr_Out2, Addr_Out1, Addr_Out0;
assign {Addr_Out4, Addr_Out3, Addr_Out2, Addr_Out1, Addr_Out0} = adrout;
```

```
reg [3:0] measured_value;
reg Measured_Level3, Measured_Level2, Measured_Level1, Measured_Level0;
assign {Measured_Level3, Measured_Level2, Measured_Level1, Measured_Level0} = measured_value;

reg [3:0] desired_value;
reg Desired_Level3, Desired_Level2, Desired_Level1, Desired_Level0;
assign {Desired_Level3, Desired_Level2, Desired_Level1, Desired_Level0} = desired_value;

reg [3:0] PGA_data;
reg PGA_Din3, PGA_Din2, PGA_Din1, PGA_Din0;
assign {PGA_Din3, PGA_Din2, PGA_Din1, PGA_Din0} = PGA_data;

reg [2:0] cntrl;
reg Control2, Control1, Control0;
assign {Control2, Control1, Control0} = cntrl;


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Stimulus starting point
//
// Initialize all the I/O first.  Then proceed with the rest of the simulation.
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

initial
begin

//Initialize the signals first
wr = 1;          rd = 1;
reset = 0;       adio = 'h0000;
ale = 0;         psen = 1;
adrout = Z8;
measured_value = 'h0;
desired_value = 'h0;
PGA_data=Z4;     cntrl = Z3;
Intrn = Z1;      Start_Conv = Z1;
Trim = Z1;       Boost = Z1;
JCEn = 1;
#100 reset = 1;  // Take the PSD out of reset after 100ns


//We are now ready to do some configuration of the PSD

//Port A configuration
//Configure Port A, pins pa4 to pa0 to output the latched address, and the rest
//of the port will output control information in MCU I/O mode.

//Writing "1F" to the Port A control register enables latched address output on
//pins pa4 to pa0, and the rest of the port to output MCU I/O.
write(Port_A_Cntl_Reg, 'h1f);

//Writing "FF" to Port A's direction register sets up Port A pins to be outputs.
write(Port_A_Dir_Reg,'hff);

//Port B configuration
//Since there is no latched address output on Port B, and its control register
//defaults to MCU I/O mode output, only the direction register needs to be setup.
//Only pins pb3 to pb0 will be outputting data, and the rest will be receiving
//input
write(Port_B_Dir_Reg,'h0f);

//All of Port C is output (with the exception of the Vstby input
write(Port_C_Dir_Reg,'hfb);

//There is only one output on Port D (RTCcs/), so the direction register is
//setup as follows:
write(Port_D_Dir_Reg,'h04);

//Set up the mask registers so that only the desired portion of the OMCs get
//written.  Only the desired value (MCELLAB[7:4]), and begin (MCELLBC7) can be
//written to.
write(Port_AB_OMC_Mask, 'h0f);
write(Port_BC_OMC_Mask, 'h7f);


//Write the EEPROM segment ees0, and the Flash segment fs1
//then read the SRAM

write('h0020,'h5a);              //write 5a to ees0
write('h5A00,'ha5);              //write a5 to fs1
read ('h07FE);                   //read the internal SRAM
```

```
// Wait, then initialize the gain to one and output the data on the pins
// pb2 to pb0.
#40 write(Port_B_Dout_Reg,'h01);


// Assume a small value for the output of the ADC since the gain is set
// to one
measured_value='h3;


// Load 5 into the desired value register
write(Port_AB_OMC, 'h50);


// Take the state machine out of the idle state and generate the ADC
// chip select.
#20 write(Port_BC_OMC, 'h80);



// Since the measured value is less than the desired one, the gain would
// be boosted after the interrupt was generated (3 cycles after the start
// of the state machine).  The MCU should increment the gain by 1 at that
// time.
#400 write(Port_B_Dout_Reg, 'h02);
#10 $finish;

end

initial
begin

// Generate a 10 MHz system clock used by the state machine, etc.
// Note the time scale is set in the psdsoft.run file.
clkin=0;
forever #100 clkin=~clkin;

end  //stimulus ends here
```

Appendix C—List of PSD813F Simulation Signals

This is a list of signals from the Explorer that can be viewed using the PSDsilosIII Data Analyzer. This list is based on the *tutor8xx.abl* file, and predefined signals. The list will vary depending on the names in your .abl file, but most of the signals will be the same. Note: to get at some of the internal PSD813F signals, you will have to click on the plus sign to expand the list. See circle below.


PSDsilos - d:\psdsoft\tutorial\tutor8xx\tutor\tutor8xx.spj - [SILOS Explorer]
File View Project Explorer Debug Window Help
Command:
All Modules
global
WSI_signals :
Scope "WSI_signals" Filter:
Addr_Out0_oe Addr_Out0 Addr_Out1_oe Addr_Out1 Addr_Out2_oe Addr_Out2 Addr_Out3_oe Addr_Out3 Addr_Out4_oe Addr_Out4 Boost_clk Boost_d Boost_oe Boost_pr Boost_re Boost Control0_oe Control0 Control1_oe Control1 Control2_oe Control2 Desired_Level0_clk Desired_Level0_d Desired_Level0_pr Desired_Level0_re Desired_Level0 Desired_Level1_clk Desired_Level1_d Desired_Level1_pr Desired_Level1_re Desired_Level1 Desired_Level2_clk Desired_Level2_d Desired_Level2_pr
Desired_Level2_re Desired_Level2 Desired_Level3_clk Desired_Level3_d Desired_Level3_pr Desired_Level3_re Desired_Level3 Intrn_clk Intrn_d Intrn_oe Intrn_pr Intrn_re Intrn JCEn_oe JCEn Measured_Level0_oe Measured_Level0 Measured_Level1_oe Measured_Level1 Measured_Level2_oe Measured_Level2 Measured_Level3_oe Measured_Level3 PGA_Din0_oe PGA_Din0 PGA_Din1_oe PGA_Din1 PGA_Din2_oe PGA_Din2 RTCcsn_oe RTCcsn STATE0_clk STATE0_pr STATE0_re STATE0_t
STATE0 STATE1_clk STATE1_pr STATE1_re STATE1_t STATE1 Start_Conv_clk Start_Conv_d Start_Conv_oe Start_Conv_pr Start_Conv_re Start_Conv Trim_clk Trim_d Trim_oe Trim_pr Trim_re Trim_0_clk Trim_0_d Trim_0_pr Trim_0_re Trim_0 Trim_1_clk Trim_1_d Trim_1_pr Trim_1_re Trim_1 Trim [7:0]VM [15:8]adioh [7:0]adiol ale_oe ale begin_cycle_clk begin_cycle_d
begin_cycle_pr begin_cycle_re begin_cycle clkin_oe clkin csiop [7:0]ctrl_a [7:0]ctrl_b [7:0]data [7:0]din_a [7:0]din_b [7:0]din_c [2:0]din_d [7:0]dirff_a [7:0]dirff_b [7:0]dirff_c [2:0]dirff_d [7:0]dout_a [7:0]dout_b [7:0]dout_c [2:0]dout_d [7:0]drive_a [7:0]drive_b [7:0]drive_c [2:0]drive_d ecsd0 ecsd1 ee_boot_oe ee_power_down [3:0]ee_protection ee_ready_busy_N ee_sdp_disable ee_sdp_enable ee_toggle eesel_f
ees0 ees1 ees2 ees3 [7:0]enable_a [7:0]enable_b [7:0]enable_c enable_data_half [7:0]f_protection flash_oe flash_polling flash_ready_busy flash_toggle flsel_f fs0 fs1 fs2 fs3 fs4 fs5 fs6 fs7 jtagsel jtag [7:0]mask_mcab [7:0]mask_mcbc meqd_clk meqd_d meqd_pr meqd_re meqd mxord3_clk mxord3_d mxord3_pr mxord3_re
mxord3 nib_a0 nib_a1 nib_b0 nib_b1 nib_c0 nib_c1 [7:0]out_mcab [7:0]out_mcbc pa0_imc pa1_imc pa2_imc pa3_imc pa4_imc pa5_imc pa6_imc pa7_imc pb0_imc pb1_imc pb2_imc pb3_imc pb3_oe pb3 pb4_imc pb5_imc pb6_imc pb7_imc pc0_imc pc1_imc pc2_imc pc3_imc pc4_imc pc5_imc pc5_oe pc5
pc6_imc pc6_oe pc6 pc7_imc pdn pgr0 pgr1 pgr2 pgr3 pgr4 pgr5 [7:0]pgr7_0 [7:0]pmmr1 [7:0]pmmr2 psel0 psel1 psen rd_bsy rd reset rs0_0_clk rs0_0_d rs0_0_pr rs0_0_re rs0_0 rs0 sram_oe swap vstby_oe vstby wr
For Help, press F1
T1 T2 Tdelta Time 3,780.0ns NUM

Any of the above signals can be dragged to the Data Analyzer window for viewing. Once there, the signals can be made into busses. For more information on the Explorer or Data Analyzer, see PSDsilosIII's on-line help, and the *PSDsilosIII User Manual*.

Below is a table that contains all of the viewable predefined signal names, along with a brief description of each. The conventions used in the table are:

- n represents a number
- x represents a letter

See the list above to determine which letters and/or numbers apply to the respective signal.

Table C1—Predefined Signal Names and their Descriptions

Signal/Bus Name	Description
VM	VM register
adioh[15:8]	Address/Data bus high byte
adiol[7:0]	Address/Data bus low byte
ctrl_x	Port x control register
data[7:0]	Non-multiplexed 8-bit data bus
din_x	Port x data in register
dirff_x	Port x direction register
dout_x	Port x data out register
drive_x	Port x drive register
ecsdn	External chip select output n
ee_boot_oe	EEPROM output enable
ee_power_down	EEPROM power down signal
ee_protection[3:0]	PSD security and EEPROM sector protection
ee_ready_busy_N	EEPROM ready/busy signal
ee_sdp_disable	EEPROM software data protection disable bit
ee_sdp_enable	EEPROM software data protection enable bit
ee_toggle	EEPROM toggle signal
eesel_f	EEPROM final chip select
enable_x	Enable to port x driver
f_protection[7:0]	Flash sector protection register (read only)
flash_oe	Flash output enable
flash_polling	Flash data polling bit
flash_ready_busy	Flash ready/busy signal
flash_toggle	Flash toggle bit
flsel_f	Flash final chip select
jtag	JTAG enable register
mask_mcab	Mask AB register outputs
mask_mcbc	Mask BC register outputs
mcellabn	Micro⇔Cell AB n output
mcellabn_clk	Output Micro⇔Cell AB n clock input
mcellabn_pr	Output Micro⇔Cell AB n preset input
mcellabn_reg	Output Micro⇔Cell AB n register input
mcellabn_re	Output Micro⇔Cell AB n reset input
mcellbcn	Output Micro⇔Cell BC n output
mcellbcn_clk	Output Micro⇔Cell BC n clock input
mcellbcn_pr	Output Micro⇔Cell BC n preset input
mcellbcn_reg	Output Micro⇔Cell BC n register input
mcellbcn_re	Output Micro⇔Cell BC n reset input
nib_xn	Product term control port x[7:4] or x[3:0] input Micro⇔Cell
out_mcab[7:0]	Output registers for Micro⇔Cell AB
out_mcbc[7:0]	Output registers for Micro⇔Cell BC
pxn	Port x, pin n
pxn_imc	Port x, Input Micro⇔Cell n
pxn_oe	Port x, output enable n product term
pdn	Power down signal
pgr7_0	Page register outputs
pmmrn	Power management mode register n
pseln	Port n peripheral select
rd_bsy	PSD internal ready/busy status signal
sram_oe	SRAM output enable signal

Appendix D—Design file for EPM7064S (U2 of Figure 1)

```
-- Title:        8xx Tutorial--Discrete Solution
-- Function:     Replacement for the programmable logic portions of the PSD8xx
-- Designed by:  Dan Harris
-- Design date:  6/15/98
-- Description:  This design shows what chip and logic would be required to replace the
--               programmable logic portions of the PSD813F1.  This chip will be responsible
--               for the following tasks:
--               * Latching the address generated by the 80C31 MCU.
--               * Decoding the address and generating internal/external chip selects.
--               * Storing control/status information in internal registers.
--               * Address translation for memory pageng.
--               * Interfacing to and controlling of the PGA in the Receiver circuit, and the ADC
--                 RTC, SRAM, EEPROM, FLASH, and MCU.
--               * Interfacing to a JTAG-compatible port for ISP.
-- Convention:   The tilde (~) is used throughout this design to indicate active low signals.

CONSTANT PAGE_REG_ADDR     = H"09E0";
CONSTANT VM_REG_ADDR       = H"09E2";
CONSTANT MCU_IO_OUT_ADDR   = H"0902";
CONSTANT DESIRED_REG_ADDR  = H"0920";
CONSTANT GAIN_REG_ADDR     = H"0901";
CONSTANT START_SIG_ADDR    = H"0921";

subdesign 8xxtutor
(
-- The following signals are generated by the MCU (U1):
A/D[7..0]         : BIDIR; -- Multiplexed address (lower byte)/data bus
A[15..8]          : INPUT; -- Upper byte of the addr bus
RD~               : INPUT; -- Read strobe
WR~               : INPUT; -- Write strobe
ALE               : INPUT; -- addr latch enable signal
PSEN~             : INPUT; -- Program store enable

-- System-level inputs:
Reset~            : INPUT; -- System reset
Clock             : INPUT; -- System clock

-- The following signals are generated for the MCU (U1):
AGC_Interrupt~    : OUTPUT;-- Interrupt the MCU when the desired and measured signal levels don't match
Trim              : OUTPUT;        -- True when the measured level is greater than the desired one
Boost             : OUTPUT;        -- Opposite of Trim

-- The chip select output for the RTC (U5):
RTC_CS~           : OUTPUT;

-- This signals are to/from the ADC (U7):
Start_Conversn    : OUTPUT; -- Indicates when the ADC should start its analog-to-digital conversion
ADC_Out[3..0]     : INPUT;  -- The measured signal strength

-- The bus is used to set the gain on the PGA (part of U8)
PGA_Din[2..0]     : OUTPUT;

-- The following are outputs to the external memories:
-- Chip selects
FLASH_CS~         : OUTPUT;
EEPROM_CS~        : OUTPUT;
SRAM_CS~          : OUTPUT;
-- Output enables
FLASH_OE~         : OUTPUT;
EEPROM_OE~        : OUTPUT;
SRAM_OE~          : OUTPUT;
-- Upper address bits
FLASH_A[16..14]   : OUTPUT;        -- MS addr bits for the 128K FLASH - for segmentation
EEPROM_A[14..13]  : OUTPUT;        -- MS addr bits for the 32K EEPROM - for segmentation

-- Latched/demultiplexed address output
Addr_Out[7..0]    : OUTPUT;        -- outputs to the external memories

-- Control Output for MCU I/O mode
Control[2..0]     : OUTPUT;
)

VARIABLE

A/D[7..0]                : TRI;   -- Needed to drive the data output onto the data bus
```

```
la[7..0]                : LATCH;-- Must demux lower byte of addr
page_reg[7..0]          : DFFE;  -- Page register
vm_reg[7..0]            : DFFE; -- Used for memory mapping in combined memory space mode
desired_reg[3..0]       : DFFE;  -- Register to store the desired signal level (set by the MCU)
gain_reg[2..0]          : DFFE; -- Register to store the gain level (set by the MCU)
begin_comparrison       : DFFE; -- takes state machine out of idle state (s0)
cntrl_port_reg[2..0]    : DFFE;  -- MCU I/O mode control register
addr[15..0]             : NODE;  -- Demultiplexed addr
fs[7..0]                : NODE;  -- FLASH segment enable signals
ees[3..0]               : NODE; -- EEPROM segment enable signals
swap                    : NODE; -- bit 7 of the page register
enable_data_half        : NODE;  -- bit 6 of the page register
measured[3..0]          : NODE;  -- Output from the ADC
desired[3..0]           : NODE;  -- Input from the MCU
meqd                    : NODE;  -- True when the measured value equals the desired one
sm                      : MACHINE WITH STATES (s0, s1, s2, s3);

BEGIN

-- Right now, there is nothing to output to the MCU on the A/D lines
  A/D[] = GND;

-- Latch in the addr[]
  la[] = A/D[];
  la[].ena = ALE;
  addr[7..0] = la[];
  addr[15..8] = A[];
  Addr_Out[] = la[];

  begin_comparrison = A/D7;
  begin_comparrison.clk = Clock;
  begin_comparrison.clrn = Reset~;
  begin_comparrison.ena = !WR~ & (addr[] == START_SIG_ADDR);

  desired_reg[] = A/D[7..4];
  desired_reg[].clk = Clock;
  desired_reg[].clrn = Reset~;
  desired_reg[].ena = !WR~ & (addr[] == DESIRED_REG_ADDR);

  gain_reg[] = A/D[2..0];
  gain_reg[].clk = Clock;
  gain_reg[].clrn = Reset~;
  gain_reg[].ena = !WR~ & (addr[] == GAIN_REG_ADDR);

  cntrl_port_reg[] = A/D[2..0];
  cntrl_port_reg[].clk = Clock;
  cntrl_port_reg[].clrn = Reset~;
  cntrl_port_reg[].ena = !WR~ & (addr[] == MCU_IO_OUT_ADDR);

  page_reg[] = A/D[];
  page_reg[].clk  = Clock;
  page_reg[].clrn = Reset~;
  page_reg[].ena  = !WR~ & (addr[] == PAGE_REG_ADDR);

  vm_reg[] = A/D[];
  vm_reg[].clk  = Clock;
  vm_reg[].clrn = Reset~;
  vm_reg[].ena  = !WR~ & (addr[] == VM_REG_ADDR);

  measured[] = ADC_Out[];
  desired[] = desired_reg[];
  PGA_Din[] = gain_reg[];
  Control[] = cntrl_port_reg[];

-- Memory Section
  swap = page_reg7;
  enable_data_half = page_reg6;

  fs0 = ((addr[] >= H"8000") & (addr[] <= H"BFFF") & (page_reg[] == 3) & !swap)
        # ((addr[] >= H"0000") & (addr[] <= H"3FFF") & swap);
  fs1 = (addr[] >= H"4000") & (addr[] <= H"7FFF");
  fs2 = (addr[] >= H"8000") & (addr[] <= H"BFFF") & (page_reg[] == 0);
  fs3 = (addr[] >= H"C000") & (addr[] <= H"FFFF") & (page_reg[] == 0);
  fs4 = (addr[] >= H"8000") & (addr[] <= H"BFFF") & (page_reg[] == 1);
  fs5 = (addr[] >= H"C000") & (addr[] <= H"FFFF") & (page_reg[] == 1);
  fs6 = (addr[] >= H"8000") & (addr[] <= H"BFFF") & (page_reg[] == 2);
  fs7 = (addr[] >= H"C000") & (addr[] <= H"FFFF") & (page_reg[] == 2);

  ees0 = ((addr[] >= H"0000") & (addr[] <= H"1FFF") & !swap)
       # ((addr[] >= H"8000") & (addr[] <= H"9FFF") &  swap & !enable_data_half);
  ees1 = ((addr[] >= H"2000") & (addr[] <= H"3FFF") & !swap)
```

```
       # ((addr[] >= H"A000") & (addr[] <= H"BFFF") &  swap & !enable_data_half);
  ees2 =  (addr[] >= H"C000") & (addr[] <= H"DFFF") &  swap &  enable_data_half;
  ees3 =  (addr[] >= H"E000") & (addr[] <= H"FFFF") &  swap &  enable_data_half;

  -- FLASH upper 3 and EEPROM upper 2 address bit encoding

  FLASH_A16 = fs7 # fs6 # fs5 # fs4;
  FLASH_A15 = fs7 # fs6 # fs3 # fs2;
  FLASH_A14 = fs7 # fs5 # fs3 # fs1;
  EEPROM_A14 = ees3 # ees2;
  EEPROM_A13 = ees3 # ees1;

-- Chip Selects and Output Enables
-- SRAM has highest priority, followed by EEPROM, and then FLASH
  !FLASH_CS~ = (fs0 # fs1 # fs2 # fs3 # fs4 # fs5 # fs6 # fs7) & (EEPROM_CS~ # SRAM_CS~);
  !EEPROM_CS~ = (ees0 # ees1 # ees2 # ees3) & SRAM_CS~;
  !SRAM_CS~ = ((addr[] >= H"0100") & (addr[] <= H"08FF"));
  !RTC_CS~ = ((addr[] >= H"0A00") & (addr[] <= H"0A1F"));

  !SRAM_OE~ = !(!RD~ # (!PSEN~ & vm_reg0));
  !EEPROM_OE~ = !((!PSEN~ & vm_reg1) # (vm_reg3 & !RD~));
  !FLASH_OE~ = !((!PSEN~ & vm_reg2) # (vm_reg4 & !RD~));

-- Comparator I/O
  Trim = (measured[] > desired[]);
  meqd = (measured[] == desired[]);
  Boost = !Trim & !meqd;


-- State Machine
  sm.clk = Clock;
  sm.reset = !Reset~;

  CASE sm IS
        WHEN s0 =>
      Start_Conversn = GND;
          AGC_Interrupt~ = VCC;
          IF (begin_comparrison) THEN
                sm = s1;
          ELSE
                sm = s0;
          END IF;
        WHEN s1 =>
          Start_Conversn = VCC;
          sm = s2;
        WHEN s2 =>
          Start_Conversn = GND;
          sm = S3;
        WHEN s3 =>
          !AGC_Interrupt~ = Trim # Boost;  -- Interrupt when Measured not equal to Desired
          sm = s0;
  END CASE;

END;
```

Appendix E—Discrete Solution (Figure 1) Compared to Integrated PSD Solution (Figure 2)

This appendix compares the two circuits of Figures 1 and 2 in the following categories: (Only the major ICs were compared.)

- Cost
- Average Current Usage
- Board Space Usage
- Time to market.

Cost

The 150 ns PSD813F1 in the PLCC package can be purchased at a significantly lower price than the total cost of the individual EEPROM, Flash, SRAM, and CPLD devices.

Average Current Usage

The PSD813F would typically use 4.88 mA[1] according a calculation based on the "Example of PSD813F Typical Power Calculation at Vcc = 5.0 V" in the "AC/DC Parameters" section of the *PSD813F Family Data Sheet*. Now, if we take the total average current of the devices in the discrete solution, we get 32.4 mA (with the EPM7064S not in turbo mode). This shows that the discrete solution uses 664% more current than the PSD!

Board Space Usage

The PSD813F1 in the PLCC package takes up 400 mm^2. The chips that make up the discrete solution take up a combined 1493 mm^2. That equates to 373% more board space! (All calculations based on PLCC packages.) This calculation does not reflect the extra board space, complexity, and noise associated with routing the signals in the discrete solution.

Time to Market

While no specific quantities can be used for a calculation, it should be obvious that the time to market will be reduced significantly for many reasons. The easiest to visualize is the fact that the discrete solution involves 4 complex ICs to deal with instead of just one. Also, there are templates and predefined routines that, when used in conjunction with our user-friendly PSDsoft, will help you with every step of your design process. Issues related to concurrent memory, memory mapping, and MCU assisted ISP are simplified. Even C code is generated for you. The JTAG interface is one of the greatest benefits and time savers; it allows you to program, configure, and test the entire PSD, and leave it soldered to the board the whole time! Last, but not least, there are just fewer places to go wrong, and fewer things to debug when you have this level of integration.

[1] Calculation based on a 16 MHz crystal input to the 80C31 MCU, 8 MHz composite input frequency to the PLD, and 66 product terms (based on output from fitter report). The rest of the parameters used for the calculation are identical to the ones used in the "Example of PSD813F Typical Power Calculation at Vcc = 5.0 V" section of the *PSD813F Family Data Sheet.*

Appendix F—System Memory Map and UART ISP

Introduction

A system memory map was developed for this tutorial to take full advantage of the memory available in the PSD813F1 and expand beyond the 64 Kbyte address space limitation of the 8031 MCU. This memory map facilitates the downloading of firmware from a host computer to the Flash memory in the PSD using the 8031 UART. The 8031 boots from the PSD EEPROM, concurrently downloads to PSD Flash memory, and then 8031 execution jumps from EEPROM to Flash memory. After this jump, the EEPROM in the boot area address space is replaced with Flash memory by a special register within the PSD (the VM Register). After that, the entire Flash memory is available to the 8031.

This system memory map also allows the concurrent downloading of boot code into the PSD EEPROM while executing code out of PSD Flash memory. This is not possible in non-PSD systems that use PROM for boot code.

The total memory available to the 8031 as defined in this system is:

- 128 Kbytes Flash
- 16 Kbytes EEPROM for boot code
- 16 Kbytes EEPROM for data storage
- 2 Kbytes battery-backed SRAM (in addition to the 128 bytes SRAM resident on the 8031)

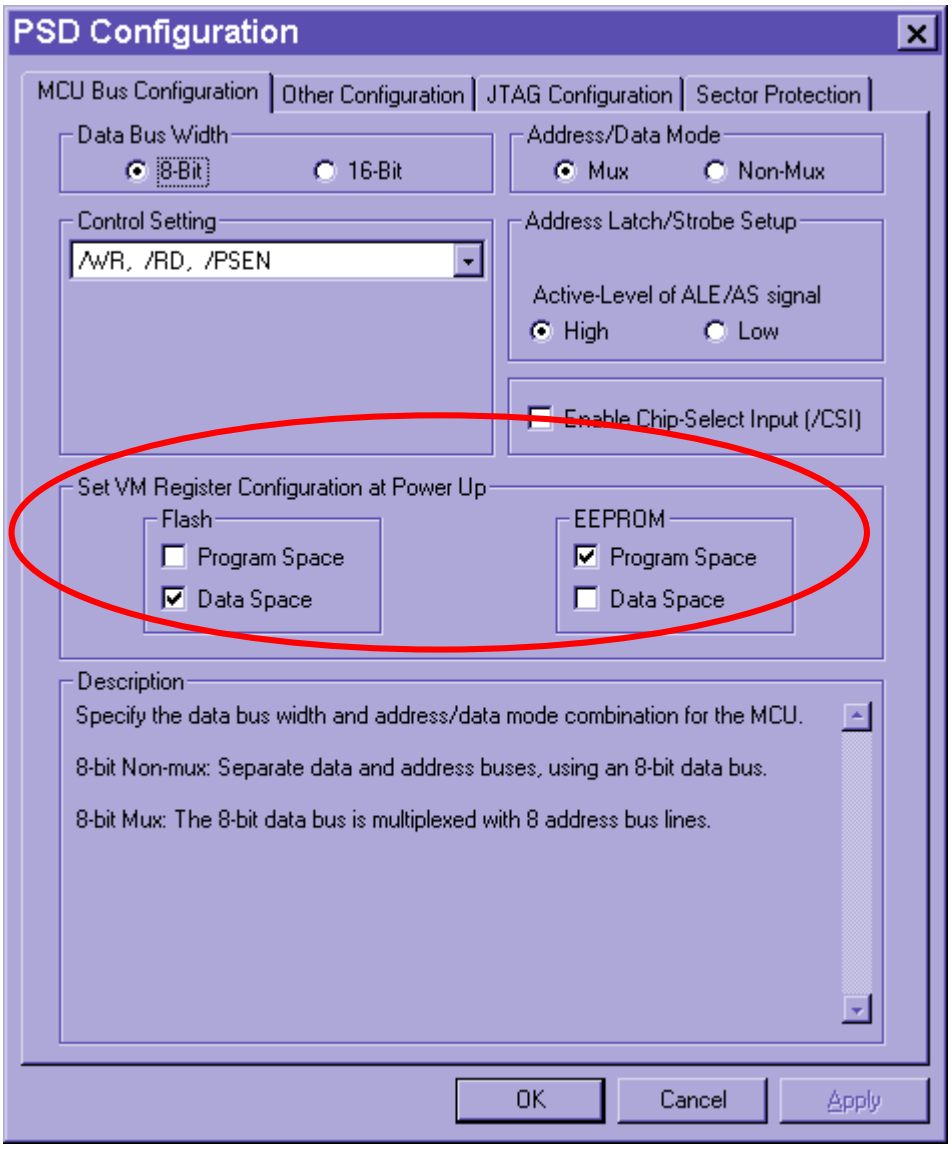
System Memory Map

The system memory map is shown in Figures F1, F2, F3, and F4. The labels FSx and EESx are the names of internal memory segments within the PSD813F1 device. FSx represents 16 Kbyte Flash segments, EESx represents 8 Kbyte EEPROM segments.

In this design, paging is used because the system contains more memory than the 8031 can address linearly. The PSD813F1 facilitates paging by using a page register, which the 8031 can access. Because paging is used, a common memory area is needed for firmware routines that must be accessible regardless of what page the MCU is executing from. This common area resides in the lower half of each memory page in program space (shown in Figures F1 through F4); it should contain routines that handle initialization, interrupts, implement page switching, and drive physical devices. It is also used to keep critical data space items available at all times. For example, in this design, the PSD control registers, I/O, and system SRAM for the stack and global variables are available on any memory page (see Figures F1 through F4).

There are two fundamental modes of operation: one is boot/download mode, and the other is normal operation. Figures F1 through F4 show the memory map during the transition from boot/download mode to normal operation mode.

Figure F1 represents the memory map at power-on (boot). The system will boot up from EEPROM, and then facilitate a download to the main Flash memory (if needed) using the 8031 UART. At this point, all of the PSD Flash memory is in 8031 "data space" and all of the EEPROM is in 8031 "program space". This is due to the "MCU Bus Configuration" that was done in step 2 of section 5.3 (PSD Configuration), and shown again, below. This step of the configuration automatically sets the VM register to 12h. Refer to the *PSD813F Family* Data Sheet for information on VM register settings.



It is very important to note that the PSD Configuration utility sets the VM register (located in CSIOP space at offset E2h) initially, and that it can only be changed by the MCU after it has booted.

After the Flash has been programmed and/or validated, the Flash memory is moved from 8031 data space to 8031 program space by the MCU writing 06h to the VM register (while still executing out of PSD EEPROM).

Figure F2 represents the memory map after the Flash has been moved to program space. This is an intermediate step that is a result of writing to the VM register.

Next, 8031 execution jumps from PSD EEPROM to PSD Flash. While executing from PSD Flash, the 8031 will set a bit in the PSD page register that we call "SWAP". Note: if the "SWAP" bit was set while executing from EEPROM, indeterminite results would occur. (It's like pulling a rug out from under yourself.) Now, the EEPROM that the MCU booted from during power-up is replaced with Flash memory that contains application vectors and code, as shown in Figure F3. The transition between the two maps of Figures F2 and F3 is under control of the 8031 by setting the "SWAP" bit inside the PSD (defined in the PSDabel (*tutor8xx.abl*) file). Again, the state of the memory map shown in Figure F3 is an intermediate step.

NOTE: Individual bits within the 8-bit PSD page register may be used for functions other than memory page definition. For example, in this tutorial, two of the eight PSD page register bits are use to define four memory pages, and one of the page register bits is used as the "SWAP" bit, described above.

Finally, while executing from PSD Flash memory, the 8031 must write 0Ch to the VM register in the PSD to move the PSD EEPROM from 8031 program space to 8031 data space. At this point, the MCU may jump to the new reset vector that resides at location 0000h in Flash segment FS0. This will finalize the memory map, as shown in Figure F4. Now, all 128 Kbytes of PSD Flash memory are in program space, with 32 Kbytes in a common area and 96 Kbytes spread across three memory pages. Also, the EEPROM is now in data space and is accessible from any memory page. Notice that two PSD EEPROM segments (EES2 and EES3) appear in Figure F4. These two segments are for general data use while the other two EEPROM segments (EES0 and EES1) comprise the 8031 power-on boot code.

Now that the system memory map looks like that of Figure F4, another feature becomes available. Besides the mechanisms mentioned, there is one more memory mapping control bit used in this tutorial design. This bit, "ENABLE_DATA_HALF", is another PSD page register bit used to protect the boot code in EES0 and EES1 from inadvertent writes. At the same time, it enables the other half of EEPROM (EES2 and EES3) to be accessed for general data. For example, to update the boot code in EES0 and EES1 with new code downloaded over the UART, the 8031 would leave ENABLE_DATA_HALF at logic zero, perform the update by writing to EES0 and EES1, then set ENABLE_DATA_HALF to logic one. Now the new boot code is inaccessible (protected while not booting), and the data half of EEPROM is accessible.

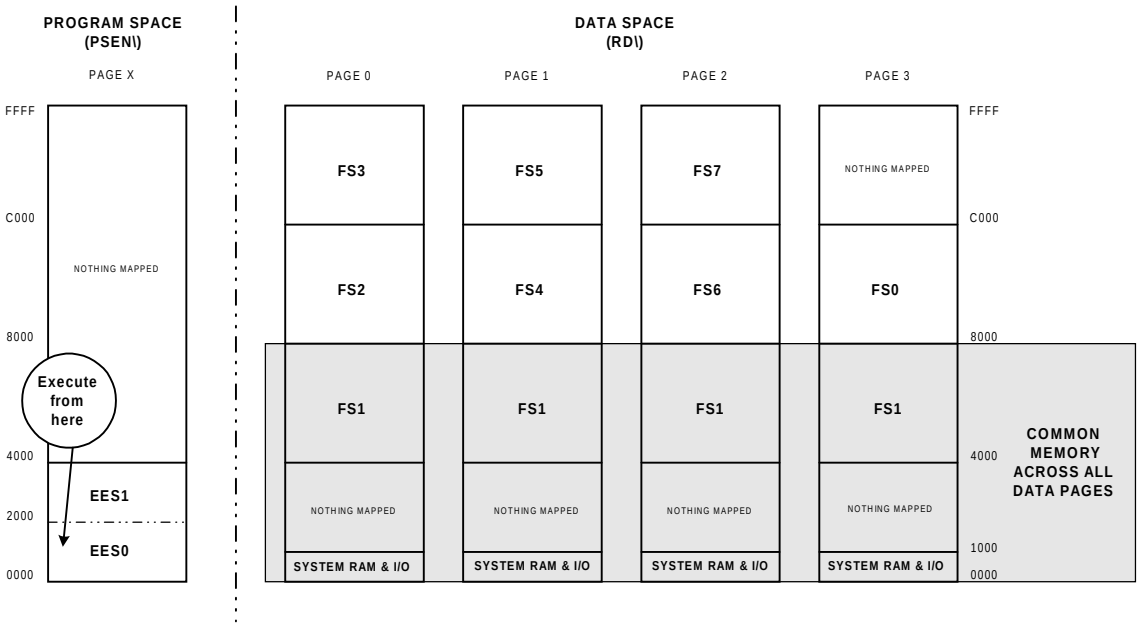


Figure F1 - System Memory Map for 8031-PSD813F1, boot/download

POWER-UP (VM Register = 12h)

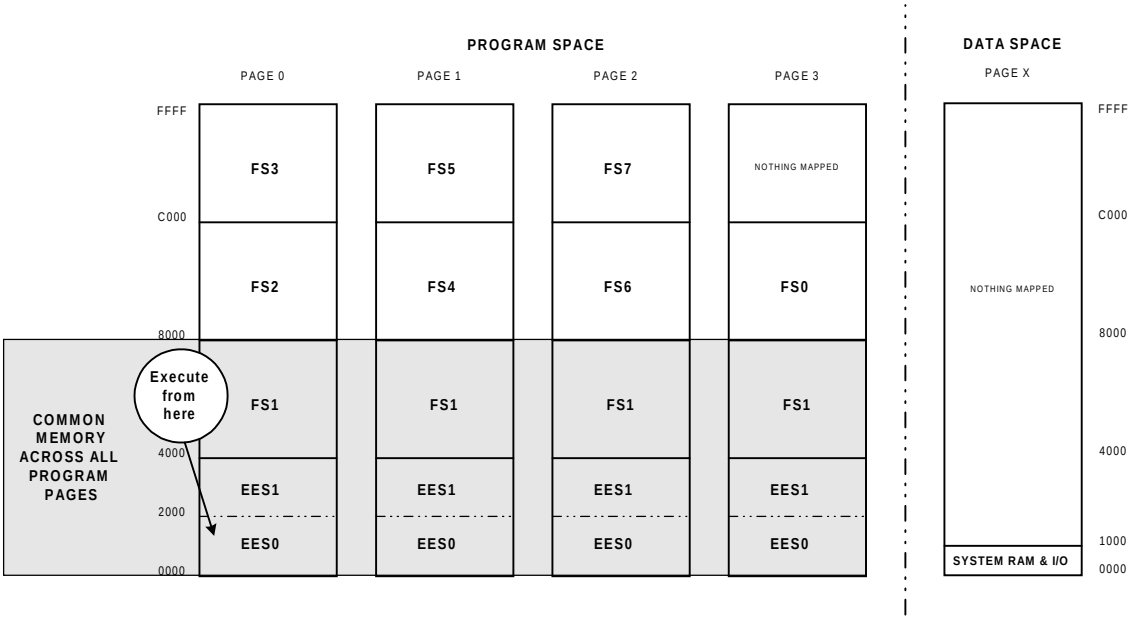
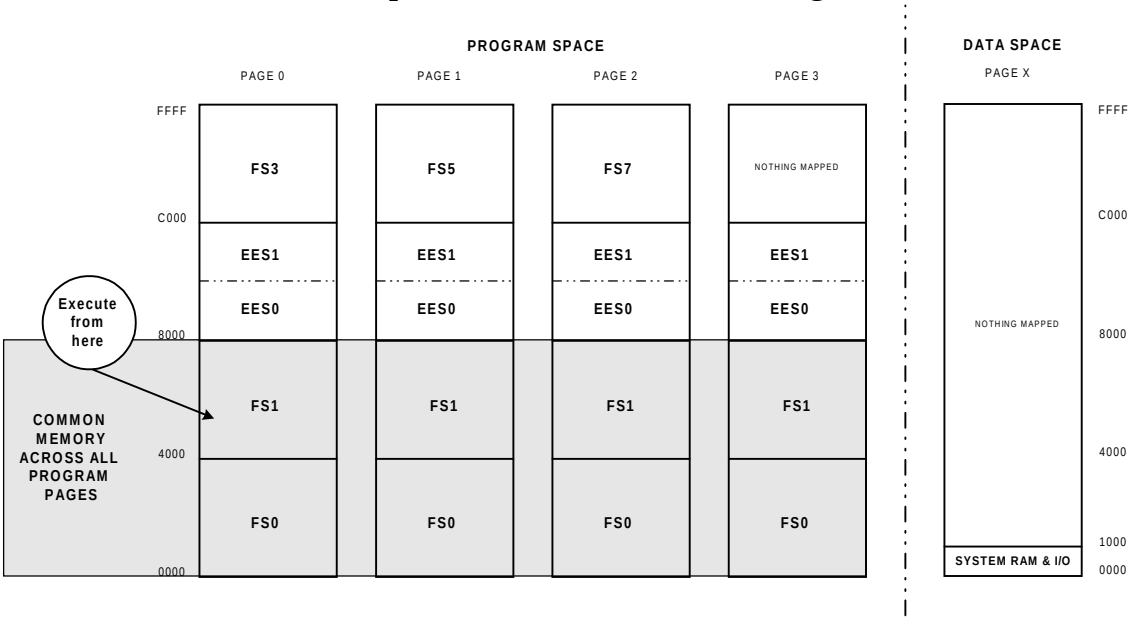


Figure F2 - System Mem Map for 8031-PSD813F1, move Flash to program space

WRITE 06h TO THE VM REGISTER

Figure F3 - System Memory Map for 8031-PSD813F1, swap boot EEPROM with Flash segment



SET SWAP BIT = 1

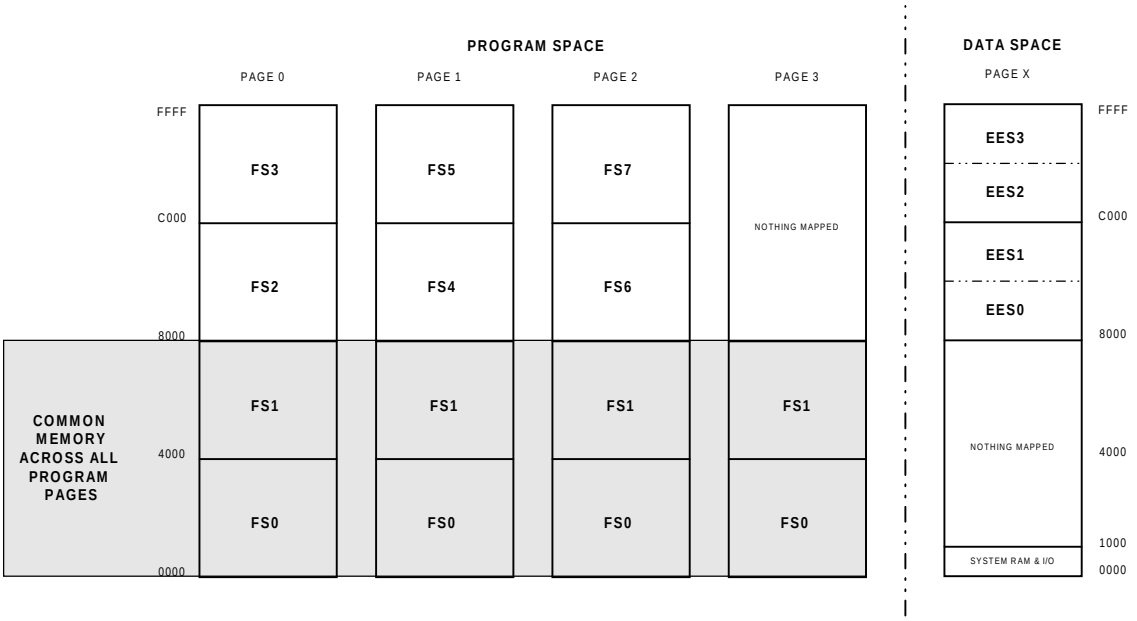


Figure F4 – Final Sys Mem Map for 8031-PSD813F1, move EEPROM to data space

WRITE 0Ch TO THE VM REGISTER

Code partitioning

Now, let's look at partitioning code in the Flash memory pages. Ultimately, the MCU will be executing from Flash memory since the EEPROM is used for boot-up and ISP in this design. Let's assume that we will have 128 Kbytes of program space in Flash memory, as shown in Figure F4. The 128 Kbytes of code will reside in four areas: 32 Kbytes in the common area (FS0 and FS1, accessible from any page), 32 Kbytes on page zero (FS2 and FS3), 32 Kbytes on page one (FS4 and FS5), and 32 Kbytes on page two (FS6 and FS7).

Keep in mind that if the 8031 never leaves page zero while executing, it can access 64 Kbytes of Flash memory in FS0 through FS3 as well as all SRAM and I/O. If the 8031 execution jumps to Flash memory on pages one or two from a call on the upper half of page zero (FS2 or FS3), care must be taken to leave a path to return to page zero again. However, if the call to page one or two was from a routine in the lower half of page zero (common area, FS0 or FS1), there is no problem returning from the call.

When placing code in Flash memory on the upper half of pages zero, one, or two, the software designer must break tasks into logical groups. These groups should not need to access code on other pages frequently. (Most software can be split in this manner and is a result of a good modular design.) Since system SRAM is available on any page, firmware routines that reside on different pages may pass data using global variables or the stack. The designer can create page-switching algorithms to jump between tasks on different pages. There are many ways to implement a paging scheme: one method involves the use of a table that contains addresses and page numbers of all program tasks, which may be called from page to page. The table and algorithms must reside in the portion of Flash memory that’s located in the common area. This provides a very clean paging solution, which may be implemented using a high-level compiler. (The compiler from Keil supports this directly, and creates the tables for you.) The only penalty when using this method is the overhead experienced when switching from one page to another.

For this tutorial design, five different files from an MCU cross-compiler and linker will be used to program the NVM memory sections of the PSD813F1. These are dummy files with no code in them, but are present to illustrate the merging of MCU firmware with the PSD configuration during the Address Translate operation. If this were a real design, the file *common.hex* would contain all of the common functions and interrupt vectors, and would be programmed into FS0/FS1. Three more files from the MCU linker, *page_0.hex, page_1.hex,* and *page_2.hex* would contain the partitioned code described above. As such, these three files would be programmed into segments FS2/FS3, FS4/FS5, and FS6/FS7, respectively. Finally, the file *boot.hex*, containing the power-up boot code and programming algorithms for Flash memory, would be programmed into EES0/EES1.

Start-up sequences, UART downloads

Let's assume that it is desired to use a PC or laptop as a host to download firmware to this embedded system over an RS-232 UART channel (instead of JTAG). These download actions can program the main Flash memory for the very first time; can update the main Flash after it has been programmed once; or can update the boot code after being programmed for the first time by a device programmer or JTAG link.

There are six valid boot-up scenarios (a through e) that must be handled by the system at power-up (reset). The default setting of the VM register at power-up places the main Flash memory in data space and the EEPROM in program space. Refer to the memory maps in Figures F1 through F4. The boot-up scenarios are as follows:

a. RS-232 cable not attached, main Flash valid.

8031 action:

- Boot from EES0/EES1
- Run a checksum on the Flash memory
- Check the UART for a pending host download request of main Flash (Figure F1)
- Set a bit in the PSD VM register to put the main Flash into program space (Figure F2)
- Set the SWAP bit in PSD, which swaps EES0/EES1 with FS0 (Figure F3)
- Set a bit in the PSD VM register to put the EEPROM into data space (Figure F4)

Now, the system is in normal operating mode. More 8031 action:

- Check the UART for a host download request of boot memory
- Set the ENABLE_DATA_HALF bit in the PSD if no boot download request exists

Normal application code can now be executed from main Flash memory.

b. RS-232 cable attached, main Flash valid, no download demands from host.

Action: same as step "a.", above.

c. RS-232 cable attached, main Flash valid, download of main Flash is demanded by host.

8031 action:

- Boots from EES0/EES1
- Run a checksum on the Flash memory
- Check the UART for a pending host download request of main Flash (Figure F1)
- Program the main Flash memory with data from the UART and run a checksum.
- Set a bit in the PSD VM register to put the main Flash into program space (Figure F2)
- Set the SWAP bit in PSD, which swaps EES0/EES1 with FS0 (Figure F3)
- Set a bit in the PSD VM register to put the EEPROM into data space (Figure F4)

Now, the system is in normal operating mode. More 8031 action:

- Check the UART for a host download request of boot memory
- Set the ENABLE_DATA_HALF bit in the PSD if no boot download request exists

Normal application code can now be executed from main Flash memory.

d. RS-232 cable not attached, main Flash is blank or invalid.

8031 action:

- Boot from EES0/EES1
- Run a checksum on the Flash memory
- Check the UART for a pending host download request of main Flash (Figure F1)
- Wait until any UART traffic is present (Figure F1)

e. RS-232 cable attached, main Flash is blank or invalid.

8031 action:

- Boot from EES0/EES1
- Run a checksum on the Flash memory
- Check the UART for a pending host download request of main Flash (Figure F1)
- Program the main Flash memory with data from the UART and run a checksum
- Set a bit in the PSD VM register to put the main Flash into program space (Figure F2)
- Set the SWAP bit in PSD, which swaps EES0/EES1 with FS0 (Figure F3)
- Set a bit in the PSD VM register to put the EEPROM into data space (Figure F4)

Now, the system is in normal operating mode. More 8031 action:

- Check the UART for a host download request of boot memory
- Set the ENABLE_DATA_HALF bit in the PSD if no boot download request exists

Normal application code can now be executed from main Flash memory.

f. RS-232 cable attached, main Flash is valid, system requests a download of boot memory

8031 action:

- Boot from EES0/EES1
- Run a checksum on the Flash memory
- Check the UART for a pending host download request of main Flash (Figure F1)
- Set a bit in the PSD VM register to put the main Flash into program space (Figure F2)
- Set the SWAP bit in PSD, which swaps EES0/EES1 with FS0 (Figure F3)
- Set a bit in the PSD VM register to put the EEPROM into data space (Figure F4)

Now, the system is in normal operating mode. More 8031 action:

- Check the UART for a host download request of boot memory
- Program the EEPROM boot memory in EES0 and EES1 with data from the UART
- Run a checksum on EES0 and EES1
- Set the ENABLE_DATA_HALF bit in the PSD to protect the boot code in EES0 and EES1 from inadvertent writes
- Enable data access of EES2 and EES3

Normal application code can now be executed from main Flash memory.

Note: for of any of these host UART download options, it is assumed that the normal boot (EES0/EES1) area was programmed the very first time by a device programmer before the PSD was installed on the circuit card or by the JTAG interface while the PSD is in-system.

Return to Main Menu